Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover in $O(\log^3 n)$ Worst Case Update Time

Sayan Bhattacharya* Monika Henzinger† Danupon Nanongkai‡

Abstract

We consider the problem of maintaining an approximately maximum (fractional) matching and an approximately minimum vertex cover in a dynamic graph. Starting with the seminal paper by Onak and Rubinfeld [STOC 2010], this problem has received significant attention in recent years. There remains, however, a polynomial gap between the best known worst case update time and the best known amortised update time for this problem, even after allowing for randomisation. Specifically, Bernstein and Stein [ICALP 2015, SODA 2016] have the best known worst case update time. They present a deterministic data structure with approximation ratio $(3/2+\epsilon)$ and worst case update time $O(m^{1/4}/\epsilon^2)$, where m is the number of edges in the graph. In recent past, Gupta and Peng [FOCS 2013] gave a deterministic data structure with approximation ratio $(1+\epsilon)$ and worst case update time $O(\sqrt{m}/\epsilon^2)$. No known randomised data structure beats the worst case update times of these two results. In contrast, the paper by Onak and Rubinfeld [STOC 2010] gave a randomised data structure with approximation ratio $O(1)$ and amortised update time $O(\log^2 n)$, where n is the number of nodes in the graph. This was later improved by Baswana, Gupta and Sen [FOCS 2011] and Solomon [FOCS 2016], leading to a randomised date structure with approximation ratio 2 and amortised update time $O(1)$.

We bridge the polynomial gap between the worst case and amortised update times for this problem, without using any randomisation. We present a deterministic data structure with approximation ratio $(2+\epsilon)$ and worst case update time $O(\log^3 n)$, for all sufficiently small constants ϵ.

1 Introduction

A matching in a graph is a set of edges that do not share any common endpoint. In the dynamic matching problem, we want to maintain an (approximately) maximum-cardinality matching when the input graph is undergoing edge insertions and deletions. The time taken to handle an edge insertion or deletion in the input graph is called the *update time* of the concerned dynamic algorithm. We want a dynamic algorithm whose update time is as small as possible.

We denote the number of nodes and edges in the input graph by n and m respectively. The value of n remains fixed over time, since the set of nodes in the graph remains the same. However, the value of m changes as edges get inserted or deleted in the graph. Similar to static problems where we want the running time of an algorithm to be polynomial in the input size, in the dynamic setting we desire the update time to be polylog(n), for an input (edge insertion or deletion) to a dynamic problem can be specified using $O(\log n)$ bits.

The dynamic matching problem has been extensively studied in the past few years. We now know that within polylog(n) update time we can maintain a 2-approximate matching using a randomized algorithm [15, 1, 13] and a $(2+\epsilon)$-approximate matching using a deterministic algorithm [5, 4, 3]. The downside of these algorithms, however, is that their update times are *amortised*. Thus, the algorithms take polylog(n) update time *on average*, but from time to time they may take as large as $O(n)$ time to respond to a single update. It is much more desirable to be able to guarantee a small update time after every update. This type of update time is called *worst-case update time*.

Unfortunately, known worst-case update time bounds for this problem are *polynomial in n*: the known algorithms take $O(n^{1.495})$ worst-case update time to maintain the value of the maximum matching exactly [14], $O(\sqrt{m}/\epsilon^2)$ time to maintain a $(1+\epsilon)$-approximate maximum matching [8, 12], $O(m^{1/3}/\epsilon^2)$ time to maintain a $(4+\epsilon)$-approximate maximum matching [4], and $O(m^{1/4}/\epsilon^2)$ time to maintain a $(3/2+\epsilon)$-approximate maximum matching in bipartite graphs [2]. There is no algorithm with polylog(n) worst-case update time even with a polylog(n) approximation ratio.

We note that the lack of a data structure with good worst-case update time is not at all specific to the prob-

*Institute of Mathematical Sciences, Chennai, India. Email: jucse.sayan@gmail.com.

†University of Vienna, Faculty of Computer Science. Email: monika.henzinger@univie.ac.at. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

‡KTH Royal Institute of Technology, Sweden. Email: danupon@gmail.com. Supported by Swedish Research Council grant 2015-04659 "Algorithms and Complexity for Dynamic Graph Problems".

lem of dynamic matching. Other fundamental dynamic graph problems, such as spanning tree, minimum spanning tree and shortest paths also suffer the same issue [7, 9, 10, 6]. One exception is the celebrated randomized algorithm with polylog(n) update time for dynamic connectivity [11].

Our result. We present a deterministic algorithm that maintains a *fractional matching*[1] and a *vertex cover*[2] whose sizes are within a $(2+\epsilon)$ factor of each other, for all sufficiently small constants ϵ. Since the size of a maximum fractional matching is at most 3/2 times the size of a maximum matching, we can also maintain a $(3+\epsilon)$-approximation to the *size* of the maximum matching in $O(\log^3 n)$ worst-case update time.

2 A high level overview of our algorithm

In this section, we present the main ideas behind our algorithm. The formal description of the algorithm and the analysis appears in subsequent sections.

Hierarchical Partition. Our algorithm builds on the ideas from a dynamic data structure of Bhattacharya, Henzinger and Italiano [4] called (α, β)-*decomposition*. This data structure maintains a $(2+\epsilon)$-approximate maximum fractional matching in $O(\log n/\epsilon^2)$ amortised update time. It defines the fractional edge weights using *levels* of nodes and edges. In particular, fix two constants $\alpha, \beta \geq 1$, and recall that the input graph $G = (V, E)$ has $|V| = n$ nodes. Partition the node set V into $L+1$ levels $\{0, \ldots, L\}$, where $L = \log_\beta n$. Let $\ell(y) \in \{0, \ldots, L\}$ denote the level of a node $y \in V$. The level of an edge (x, y) is given by Eq. (2.1), and we assign a fractional weight $w(x, y)$ as per Eq. (2.2).

$$\ell(x, y) = \max(\ell(x), \ell(y)) \tag{2.1}$$

$$w(x, y) = \beta^{-\ell(x,y)} \tag{2.2}$$

Thus, the weight of an edge decreases exponentially with its level. The weight of a node $y \in V$ is defined as $W_y = \sum_{(x,y)\in E} w(x, y)$. This equals the sum of the weights of the edges incident on it. The goal is to maintain a partition satisfying the following property.

PROPERTY 2.1. *Every node y with $\ell(y) > 0$ has weight $1/(\alpha\beta) \leq W_y < 1$. Furthermore, every node y with $\ell(y) = 0$ has weight $0 \leq W_y < 1$.*

To provide some intuition, we show how to construct a hierarchical partition satisfying Property 2.1 in the static setting, when there is no edge insertions/deletions. For notational convenience, we define $V_L^* = V$. Initially, we put all the nodes in level L, and as per equations 2.1, 2.2 we assign a weight $w(x, y) = \beta^{-L} = 1/n$ to every edge $(x, y) \in E$. Since every node has degree at most $n-1$, we get $0 \leq W_y < 1$ for all $y \in V_L^*$. We now execute a FOR loop as follows.

FOR $i = L$ to 1:

We partition the node-set V_i^* into two subsets: $V_i = \{y \in V : 1/\beta \leq W_y < 1\}$ and $V_{i-1}^* = \{y \in V : 0 \leq W_y < 1/\beta\}$. Next, we move down the nodes in V_{i-1}^* to level $i-1$. The level and weight of every edge incident on a node in $V \setminus V_{i-1}^* = V_i \cup \ldots \cup V_L$ remain unchanged during this step, as per equations 2.1 and 2.2. Hence, just after the nodes in V_{i-1}^* are moved down to level $i-1$, we get $1/\beta \leq W_y < 1$ for all nodes y at level i. The weights of the remaining edges (whose both endpoints lie in V_{i-1}^*) increase by a factor of β. Hence, the weights of the nodes in V_{i-1}^* also increase by at most a factor of β. Before the nodes in V_{i-1}^* were moved down to level $i-1$, we had $0 \leq W_y < 1/\beta$ for all $y \in V_{i-1}^*$. Thus, just after the nodes in V_{i-1}^* are moved down to level $i-1$, we get $0 \leq W_y < 1$ for all $y \in V_{i-1}^*$.

When the above FOR loop terminates, we have $1/\beta \leq W_y < 1$ for all nodes $y \in V$ at levels $\ell(y) > 0$, and $0 \leq W_y < 1$ for all nodes $y \in V$ at level $\ell(y) = 0$. Specifically, Property 2.1 is satisfied with $\alpha = 1$.

THEOREM 2.1. ([4]) *Under Property 2.1, the edge-weights $\{w(e)\}$ form a $2\alpha\beta$-approximate maximum fractional matching in G.*

In [4], Bhattacharya et al. showed that we can dynamically maintain such a partition with $\alpha = \beta = (1+\epsilon)$ in $O(\log n/\epsilon^2)$ amortised update time. The main idea is as follows. Assume that we have a partition that satisfies Property 2.1. Now an edge (u, v) is inserted or deleted. This causes W_u and W_v to increase or decrease. Hence, it might happen that some node $x \in \{u, v\}$ violates Property 2.1 after the insertion/deletion of the edge (u, v), i.e. either (1) $W_x \geq 1$ or (2) $W_x < 1/(\alpha\beta)$ and $\ell(x) > 0$. We call such a node x *dirty*, and deal with this event by changing the level of x in a straightforward way as per Figure 1: If W_x is too large (resp. too small), then we increase (resp. decrease) $\ell(x)$ by one. This causes the weights of some edges incident on x to decrease (resp. increase), which in turn decreases (resp. increases) the value of W_x. For each level $i \in [0, L]$, we define the set of edges $E_i(x)$ as follows.

$$E_i(x) = \{(x, y) \in E \mid \ell(x, y) = i\}. \tag{2.3}$$

An important observation is that as a node x moves up (resp. down) from level i to level $i+1$ (resp. $i-1$),

[1] In a fractional matching each edge is assigned a nonzero weight, ensuring that for every node the sum of the weights of the edges incident to it is at most 1. The size of a fractional matching is the sum of the weights of all the edges in the graph.

[2] A vertex cover is a set of nodes such that every edge in the graph has at least one endpoint in that set.

the edges whose weights get changed all belong to the set $E_i(x)$. Since the relevant data structures can be maintained efficiently, this implies that the runtime of one iteration of the WHILE loop in Figure 1 is dominated by the cost of Line 7, which takes $O(|E_i(x)|)$ time. In [4], the authors showed that this cost can be amortised over previous edge insertions/deletions.

Note that one iteration of the WHILE loop can make some neighbours of x dirty, and x itself might remain dirty at the end of the iteration. These dirty nodes are dealt with in subsequent iterations in a similar way (until there is no dirty node left).

```
01. WHILE there is a dirty node x
02.     Let i = ℓ(x)
03.     IF W_x ≥ 1, THEN   // In this case i < L
04.         Set ℓ(x) ← ℓ(x) + 1.
05.     ELSE   // In this case W_x < 1/(αβ), i > 0
06.         Set ℓ(x) ← ℓ(x) − 1.
07.     Update ℓ(x, y) for all (x, y) ∈ E_i(x).
```

Figure 1: Fixing the dirty nodes.

Example: Inserting edges to a star. The following example shows the basic idea behind the amortisation argument. Consider a star centred at node v consisting of β^{i-1} edges, for some large i. To satisfy Property 2.1, we can set $\ell(v) = i$, while all other nodes have level 0. Thus, we get $W_v = 1/\beta$ since every edge has weight $1/\beta^i$. Now keep inserting edges to the star (the graph remains a star throughout). Property 2.1 remains satisfied until the $(\beta^i - \beta^{i-1})$-th edge is inserted – at this point the star consists of β^i edges, $W_v = 1$, and the node v becomes dirty. We fix the node by increasing $\ell(v)$ to $i+1$ as in Algorithm 1, thus reducing the edge-weights to $1/\beta^{i+1}$ and the value of W_v to $1/\beta$. To do this we have to pay the cost of $O(|E_i(v)|) = O(\beta^i)$ in terms of update time. We can amortise this cost over the $(\beta^i - \beta^{i-1})$ newly inserted edges. This gives an amortised update time of $O(1)$ for constant β.

Note that in the above example the algorithm does not perform well in the worst case: after the $(\beta^i - \beta^{i-1})$-th insertion it has to "probe" all edges in $E_i(v)$. So the worst case update time becomes $O(\beta^i)$, which can be polynomial in n when i is large. But in this particular instance the problem can be fixed easily: Whenever v becomes dirty due to the insertion of an edge with weight $1/\beta^i$, we reduce the weight of the newly inserted edge and some other edge in the star from $1/\beta^i$ to $1/\beta^{i+1}$. Thus, the net increase in the weight of v becomes equal to $1/\beta^i - 2(1/\beta^i - 1/\beta^{i+1}) = 2/\beta^{i+1} - 1/\beta^i \leq 0$ (the last inequality holds as long as $\beta \geq 2$). In other words, when the node v becomes dirty, by reducing the weights of two edges to $1/\beta^{i+1}$ we can ensure that W_v again becomes smaller than one. Once every edge has weight $1/\beta^{i+1}$, we set $\ell(v) = i+1$.

Shadow-level ($\ell_y(x,y)$). To make the above idea concrete, we introduce the notion of a *shadow-level*. For every node $y \in V$ and every incident edge $(x,y) \in E$, we define the *shadow-level of y with respect to (x,y)*, denoted by $\ell_y(x,y) \in \{0,\ldots,L\}$, to be an integer in $\{0,\ldots,L\}$ such that the following property holds.

PROPERTY 2.2. *For every node y and edge (x,y), $\ell(y) - 1 \leq \ell_y(x,y) \leq \ell(y) + 1$.*

We modify the definition of the level of an edge $(x,y) \in E$ (in Eq. (2.1)) to

$$\ell(x,y) = \max(\ell_x(x,y), \ell_y(x,y)). \tag{2.4}$$

This affects the value of $w(x,y)$ and the set $E_i(y)$ as they depend on the levels of edges (see Eq. (2.2) and (2.3)). The idea of the shadow-level is that if $\ell_y(x,y) > \ell(y)$ (respectively $\ell_y(x,y) < \ell(y)$), then from the perspective of the edge (x,y) we have already increased (resp. decreased) $\ell(y)$; thus, the level and weight of (x,y) has changed accordingly. In this case, we say that y *up-marks* (respectively *down-marks*) the edge (x,y). We will use this operation when W_y is too large (resp. too small). Intuitively, y should not up-mark and down-mark edges at the same time. In particular, let $M_{down}(y) = \{(x,y) \in E : \ell_y(x,y) = \ell(y) - 1\}$ and $M_{up}(y) = \{(x,y) \in E : \ell_y(x,y) = \ell(y) + 1\}$ respectively denote the set of all edges down-marked and up-marked by y. Then, we will maintain the following property.

PROPERTY 2.3. *Either $M_{up}(y) = \emptyset$ or $M_{down}(y) = \emptyset$.*

To see the usefulness of this new definition, consider the following algorithm for dealing with the case where the graph is always a star centred at v: If there are only edge insertions, then v up-marks the newly inserted edge and another edge in $E_{\ell(v)}(v)$ whenever it becomes dirty (i.e. $W_v \geq 1$). It is easy to see that this will be enough to keep $W_v < 1$ as long as $\beta \geq 2$. Once $E_{\ell(v)}(v) = \emptyset$, we increase $\ell(v)$ by one. Similarly, if there are only edge deletions, then v can down-mark an edge in $E_{\ell(v)}(v)$ whenever it becomes dirty (i.e. $W_v < 1/(\alpha\beta)$).

The algorithm follows the same strategy when there are both edge insertions and deletions, albeit with one caveat: To ensure that Property 2.3 holds, it cannot up-mark an edge if $M_{down}(v) \neq \emptyset$, and cannot down-mark an edge if $M_{up}(v) \neq \emptyset$. Suppose that $M_{down}(v) \neq \emptyset$ and we want to reduce the weight of v. In this event, the node v picks an edge (u,v) in $M_{down}(v)$ and sets $\ell_v(u,v)$ back from $\ell(v) - 1$ to $\ell(v)$, which reduces the value of $w(u,v)$. This causes the edge (u,v) to be removed from

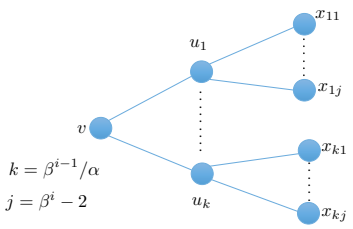

Figure 2: Example. Suppose that i is very large, e.g. $i = (\log n)/2$, and β is a large constant.

$M_{down}(v)$ and be added to $E_{\ell(v)}(v)$. We say that the node v *un-marks* the edge (u, v). Next, suppose that $M_{up}(v) \neq \emptyset$ and we want to increase the weight of v. In this event, the node v un-marks an edge in $M_{up}(v)$.

Failures. So far we have described an idea that leads to small worst-case update time when the input instance is a star graph. To make this idea work on a general input instance, we have to deal with several issues that make the algorithm more complicated. The chief one among them is the observation that the algorithm may *fail* in adjusting edge weights. Consider, for example, a tree rooted at a node v having $k = \beta^{i-1}/\alpha$ children, say $u_1, \ldots u_k$. Further, each u_i has $j = \beta^i - 2$ children. See Fig. 2. Suppose that we satisfy Property 2.1 by setting $\ell(x_{p,q}) \leftarrow 0$ for every leaf-node $x_{p,q}$ with $p \in [k]$ and $q \in [j]$, $\ell(u_p) \leftarrow i$ for every internal node u_p with $p \in [k]$, and $\ell(v) \leftarrow i$ for the root node v. No edge is down-marked or up-marked by any node. This implies that $W_v = 1/(\alpha\beta)$, $W_{u_p} = 1 - 1/\beta^i$ for all $p \in [k]$, and $W_{x_{p,q}} = 1/\beta^i$ for all $p \in [k]$ and $q \in [j]$.

Now, suppose that the edge (v, u_1) gets deleted. This makes v dirty, for W_v becomes smaller than $1/(\alpha\beta)$. The algorithm responds by down-marking an edge in $E_i(v)$, say (v, u_2). Unfortunately, this down-marking does not change the weight $w(u, v_2)$ since $\ell_{v_2}(u, v_2) = i$. I We say that this down-marking *fails*. When a down-marking fails, the node v remains dirty and Property 2.1 remains unsatisfied. In fact, in this example, the node v will remain dirty even if we down-mark all the edges in $E_i(v)$. To satisfy Property 2.1, we have no other option but to set $\ell(v) \leftarrow 0$. However, we cannot do so unless we probe all the edges in $E_i(v)$, for we have to ensure that all the down-markings on these edges fail. This takes too much time.

To deal with this issue, we keep down-marking the edges in $E_i(v)$ as long as we fail, until the point when we experience polylog(n) failures. We might still end up having $W_v < 1/(\alpha\beta)$. Nevertheless, we will guarantee a constant approximation ratio by arguing that we continue to have $W_v = \Omega(1/(\alpha\beta))$. Intuitively, every time W_v decreases by $1/\beta^i$ because of these failures, we down-mark many edges in the set $E_i(v)$. Since $|E_i(v)| \leq \beta^{i-1}/\alpha$, we are able to down-mark all the edges in $E_i(v)$ before the value of W_v becomes too small. At that point, we are ready to decrease the level of v.

Specifically, suppose that the edges incident to v keep getting deleted. While handling $t = \beta^{i-1}/(\alpha \log^2 n)$ such deletions, we perform $t \cdot \text{polylog}(n) \geq \beta^{i-1}/\alpha$ *failed* down-markings. This is enough to down-mark every edge in $E_i(v)$. At this point we move v down to level $(i-1)$, and we still have $W_v \geq 1/(\alpha\beta) - t/\beta^i = (1 - 1/\log^2 n)/(\alpha\beta)$. By repeating this argument, we conclude that even if there are more deletions, we still have $W_v = \Omega(1/(\alpha\beta))$ until the point in time when v moves down to level 0 (where v can not be dirty for its weight being small).

The algorithm in a nutshell. Our algorithm obeys the following principles. When a node y becomes dirty, it either (1) up-marks or down-marks an edge (x, y), or (2) un-marks an edge (x, y) if up-marking or down-marking would violate Property 2.3. Such an action may fail, meaning that $w(x, y)$ might not change, for reasons exemplified in Fig. 2. In this event, y continues probing its other incident edges, and stops when it experiences either its first *success* or its polylog$(n)^{th}$ failure. We show: (a) the failures do not cause the weight W_y to become too small or too large, (b) fixing one dirty node leads to at most one new dirty node, and (c) the level of the dirty node under consideration drops after constantly many fixes. Item (a) guarantees a constant approximation factor. Items (b), (c) guarantee a polylog(n) update time, for there are $O(\log n)$ levels.

3 Preliminaries

Henceforth, we focus on formally describing our dynamic algorithm and analysing its worst-case update time. For the rest of the paper, we fix two constants β, K and define L and $f(\beta)$ as in equation 3.5. Note that $K < L$ when n is sufficiently large.

$$\beta \geq 5, K = 20, f(\beta) = 1 - 3/\beta, L = \lceil \log_\beta n \rceil. \tag{3.5}$$

We will maintain a hierarchical partition of the node-set in $G = (V, E)$, and a fractional matching where the weights assigned to the edges depend on the levels of their endpoints. For technical reasons, however, there will be two key differences between the hierarchical partition actually used by our dynamic algorithm and the one that was defined in Section 2.

1. We will collapse all the nodes in levels $\{0, \ldots, K\}$ into a single level K. Accordingly, the level of a node will lie in the range $[K, L]$ in this new hierarchical partition. The weights of the nodes y in levels $\ell(y) > K$ will satisfy the constraint: $f(\beta) \leq W_y < 1$. On the other hand, the

weights of the nodes y at the lowest level $\ell(y) = K$ will satisfy the constraint: $0 \leq W_y < 1$. Comparing these constraints with Property 2.1, it follows that the term $1/(\alpha\beta)$ is replaced by $f(\beta)$ in the new partition.

2. We will allow the weight of an edge (u, v) to be off by a factor of β from its ideal value $\beta^{-\max(\ell(u),\ell(v))}$.

The structure maintained by our algorithm will be called a *nice-partition*. This is formally defined below.

DEFINITION 3.1. *In a* nice-partition*, the node-set V is partitioned into $(L - K + 1)$ subsets $V_K, \ldots, V_L$. For $i \in [K, L]$, if a node v belongs to V_i, then we say that the node v is at* level $\ell(v) = i$*. Each edge $(u, v) \in E$ gets a weight $w(u, v)$. Let $W_v = \sum_{(u,v)\in E} w(u, v)$ be the total weight received by a node v from its incident edges. The following properties hold.*

1. *For every edge $(u, v) \in E$, we have $\beta^{-\max(\ell(u),\ell(v))-1} \leq w(u, v) \leq \beta^{-\max(\ell(u),\ell(v))+1}$.*
2. *If a node v has $\ell(v) > K$, then $f(\beta) \leq W_v < 1$.*
3. *If a node v has $\ell(v) = K$, then $W_v < 1$.*

LEMMA 3.1. *Suppose that we can maintain a nice-partition in $O(T(n))$ worst-case update time. Then we can also maintain a $2/f(\beta)$-approximate maximum fractional matching and a $2/f(\beta)$-approximate minimum vertex cover in $O(T(n))$ worst case update time.*

Proof. Let $E^r = \{(u, v) \in E : \ell(u) = \ell(v) = K\}$ be the subset of edges with both endpoints at level K. We will maintain a *residual* weight $w^r(e) \geq 0$ for every edge $e \in E^r$. For notational consistency, we define $w^r(e) = 0$ for every edge $e \in E \setminus E^r$. Let $W_v^r = \sum_{(u,v)\in E^r} w^r(u, v)$ denote the residual weight received by a node v from all its incident edges. Two conditions are satisfied:

(a) For each node $v \in V$, we have $0 \leq W_v + W_v^r \leq 1$.

(b) For every edge $(u, v) \in E^r$, we have either $W_v + W_v^r \geq 1 - 1/\beta$ or $W_u + W_u^r \geq 1 - 1/\beta$.

Let $\deg^r(v)$ denote the degree of a node $v \in V$ among the edges in E^r. By condition (1) of Definition 3.1, every edge $(x, y) \in E^r$ has weight $w(x, y) \geq \beta^{-K-1}$. Hence, for every node $v \in V$, we get: $1 > W_v \geq \sum_{(u,v)\in E^r} w(u, v) \geq \deg^r(v) \cdot \beta^{-K-1}$. This implies that $\deg^r(v) < \beta^{K+1}$ for every node $v \in V$. Since β, K are constants, we get: $\deg^r(v) = O(1)$ for every node $v \in V$.

Maintaining the residual weights $\{w^r(e)\}, e \in E^r$.
For every node $v \in V$, let $b(v) = 1 - W_v$ denote the *capacity* of the node. Let $b^r(v)$ be equal to the value of $b(v)$ rounded down to the nearest multiple of $1/\beta$. We say that $b^r(v)$ is the *residual capacity* of node v. We create an *auxiliary graph* $G^* = (V^*, E^*)$, where we have β *copies* of each node $v \in V$. For every edge $(u, v) \in E^r$, there are β^2 edges in G^*: one for each pair of copies of u and v. For each node $v \in V$, if $b^r(v) = t/\beta$ for some integer $t \in [0, \beta]$, then t copies of v are *turned on* in G^*, and the remaining $(\beta - t)$ copies of v are *turned off* in G^*. We maintain a maximal matching M^* in the subgraph of G^* induced by the copies of nodes that are turned on. Since $\deg^r(v) = O(1)$ for every node $v \in V$, we can maintain the matching M^* in $O(1)$ update time using a trivial algorithm. From the matching M^*, we get back the residual weights $\{w^r(e)\}$ as follows. For every edge $(u, v) \in E^r$, if there are t edges in M^* between different copies of u and v, then we set $w^r(u, v) \leftarrow t/\beta$. It is easy to check that this satisfies both conditions (a) and (b).

Approximation guarantee.
Condition (a) implies that the edge-weights $\{w(e) + w^r(e)\}$ form a valid fractional matching in G. Define the subset of nodes $V^* = \{v \in V : W_v + W_v^r \geq f(\beta)\}$. Consider any edge $(u, v) \in E$. If at least one endpoint $x \in \{u, v\}$ lies at a level $\ell(x) > K$, then condition (2) of Definition 3.1 implies that $W_x + W_x^r \geq W_x \geq f(\beta)$, and hence $x \in V^*$. On the other hand, if both the endpoints $\{u, v\}$ lie at level K, then by conditions (a) and (b) we have: $W_x + W_x^r \geq 1 - 1/\beta \geq f(\beta)$ for some $x \in \{u, v\}$, and hence $x \in V^*$. It follows that V^* forms a valid vertex cover in G. Applying complementary slackness conditions, we infer that the edge-weights $\{w(e) + w^r(e)\}$ form a $2/f(\beta)$-approximate maximum fractional matching in G, and that V^* forms a $2/f(\beta)$-approximate minimum vertex cover in G.

Fix any constant $0 < \epsilon < 1$ and let $\beta = 3(2 + \epsilon)/\epsilon$. Then $\beta \geq 5$ and $2/f(\beta) = 2 + \epsilon$ (see equation 3.5). Setting β in this way, we can use Theorem 3.1 and Lemma 3.1 to maintain a $(2+\epsilon)$-approximate maximum fractional matching and a $(2+\epsilon)$-approximate minimum vertex cover in $O(\log^3 n)$ worst-case update time. We devote the rest of the paper to proving Theorem 3.1.

THEOREM 3.1. *We can maintain a nice-partition in $G = (V, E)$ in $O(\log^3 n)$ worst case update time.*

3.1 Shadow-levels. As in Section 2, the shadow-levels will uniquely determine the weight $w(u, v)$ assigned to every edge $(u, v) \in E$. They will ensure that $w(u, v)$ differs from the ideal value $\beta^{-\max(\ell(u),\ell(v))}$ by at most a factor of β. This implies condition (1) of Definition 3.1. Specifically, we require that each edge has two shadow-levels: one for each of its endpoints. Let $\ell_y(x, y) \in [K, L]$ be the shadow-level of a node y with respect to the edge (x, y). We require that this shadow-level can differ from the actual level of the node by at most one. This is formally stated in the invariant below.

INVARIANT 1. *For every node $y \in V$ and every edge $(x, y) \in E$, we have $\ell(y) - 1 \leq \ell_y(x, y) \leq \ell(y) + 1$.*

Next, as in Section 2, we define the *level of an edge* to be the maximum value among the shadow-levels of its endpoints. Let $\ell(x,y) \in [K,L]$ be the level of an edge (x,y). Then for every edge $(x,y) \in E$ we have:

$$\ell(x,y) = \max(\ell_x(x,y), \ell_y(x,y)). \tag{3.6}$$

As in Section 2, we now require that the weight assigned to an edge $(u,v) \in E$ be given by $\beta^{-\ell(u,v)}$.

$$w(x,y) = \beta^{-\ell(x,y)} \text{ for every edge } (u,v) \in E. \tag{3.7}$$

Thus, the weight of an edge decreases exponentially with its level. It is easy to check that if Invariant 1 holds, then assigning the weights to the edges in this manner satisfies condition (1) of Definition 3.1.

COROLLARY 3.1. *Suppose that Invariant 1 holds and edges are assigned weights as in equations 3.6, 3.7. Then for every edge $(x,y) \in E$ we have:*

$$\beta^{-\max(\ell(x),\ell(y))-1} \leq w(x,y) \leq \beta^{-\max(\ell(x),\ell(y))+1}.$$

Proof. Since each shadow-level differs from the actual level by at most one (see Invariant 1), the maximum value among the shadow-levels also differs from the maximum value among the actual levels by at most one. Specifically, we get: $\max(\ell(x),\ell(y)) - 1 \leq \ell(x,y) = \max(\ell_x(x,y),\ell_y(x,y)) \leq \max(\ell(x),\ell(y)) + 1$. The corollary now follows from the fact that the weight of an edge $(x,y) \in E$ is given by $w(x,y) = \beta^{-\ell(x,y)}$.

As in Section 2, we now define the concept of an edge *marked* by a node. Consider any edge $(x,y) \in E$ incident to a node $y \in V$. If $\ell_y(x,y) = \ell(y)+1$, then we say that the edge (x,y) has been *up-marked* by the node y. Similarly, if $\ell_y(x,y) = \ell(y) - 1$, then we say that the edge (x,y) has been down-marked by the node y. And if $\ell_y(x,y) = \ell(y)$, then we say that the edge (x,y) is *un-marked* by the node y. We let $M_{up}(y)$ and $M_{down}(y)$ respectively be the set of all edges $(x,y) \in E$ incident to y that have been up-marked and down-marked by y. For every $i \in [K,L]$, we let $E_i(y)$ be the set of all edges $(x,y) \in E$ incident to y that are at level $\ell(x,y) = i$.

$$M_{up}(y) = \{(x,y) \in E : \ell_y(x,y) = \ell(y)+1\} \tag{3.8}$$

$$M_{down}(y) = \{(x,y) \in E : \ell_y(x,y) = \ell(y)-1\} \tag{3.9}$$

$$E_i(y) = \{(x,y) \in E : \ell(x,y) = i\} \tag{3.10}$$

3.2 Different states of a node. Our goal is to maintain a nice-partition in G. In Section 3.1, we defined the concept of shadow-levels so as to ensure that the edge-weights satisfy condition (1) of Definition 3.1. In this section, we present a framework which will ensure that the *node-weights* satisfy the remaining conditions (2), (3) of Definition 3.1. Towards this end, we first need to define the concept of an *activation* of a node.

Activations of a node. The deletion of an edge (x,y) in G leads to a decrease in the values of W_x and W_y. In contrast, when an edge (x,y) is inserted in G, we assign values to its two shadow-levels $\ell_x(x,y)$ and $\ell_y(x,y)$ in such a way that Invariant 1 holds, and then assign a weight to the edge as per equation 3.7. This leads to an increase in the values of W_x and W_y. These two events are called *natural activations* of the endpoints x,y. In other words, a node is *naturally activated* whenever an edge incident to it is either inserted into or deleted from G. The weight of a node changes whenever it encounters a natural activation. Hence, such an event might lead to a scenario where the node-weight becomes either too large or too small, thereby violating either condition (2) or condition (3) of Definition 3.1. For example, consider a node y at a level $\ell(y) > K$ whose current weight is just slightly smaller than one. Thus, we have: $1-\delta \leq W_y < 1$ for some small δ. Now, suppose that y gets naturally activated due to the insertion of an edge (x,y). Further, suppose that this leads to the value of W_y becoming larger than one after the natural activation. So the node y violates condition (2) of Definition 3.1. In our algorithm, at this stage the node y will select some edge $(x',y) \in E_{\ell(y)}(y)$ and up-mark that edge. Specifically, the node will set $\ell_y(x',y) \leftarrow \ell(y)+1$, insert the edge (x',y) into the sets $M_{up}(y)$ and $E_{\ell(y)+1}(y)$, and remove the edge from the set $E_{\ell(y)}(y)$. The new level of the edge will be given by $\ell(x',y) = \ell(y)+1$. This will reduce the node-weight W_y by $\beta^{-\ell(y)} - \beta^{-(\ell(y)+1)}$, and (hopefully) the new value of W_y will again be smaller than one. The up-marking of the edge (x',y), however, will change the weight of the other endpoint x'. We call such an event an *induced activation* of x'. Specifically, an *induced activation* of a node x' refers to the event when the node-weight $W_{x'}$ increases (resp. decreases) because the other endpoint y of an incident edge (x',y) has decreased (resp. increased) its shadow-level $\ell_y(x',y)$.

In general, consider an activation of a node y that increases its weight. Suppose that the node wants to revert this change (weight increase) so as to ensure that conditions (2) and (3) of Definition 3.1 remain satisfied. Then it either up-marks some edges from $E_{\ell(y)}(y)$ or un-marks some edges from $M_{down}(y)$. This, in turn, might activate some of the neighbours of y.

Similarly, consider an activation of a node y that decreases its weight. Suppose that the node wants to revert this change (weight decrease) so as to ensure that conditions (2) and (3) of Definition 3.1 remain satisfied. Then it either down-marks some edges from $E_{\ell(y)}(y)$ or un-marks some edges from $M_{up}(y)$. Again, this might in turn activate some of the neighbours of y.

We require that a node cannot simultaneously have an up-marked and a down-marked edge incident on it. This requirement is formally stated in Invariant 2. Intuitively, a node has up-marked incident edges when it is trying to ensure that its weight does not become too large, and down-marked incident edges when it is trying to ensure that its weight does not become too small. Thus, it makes sense to assume that a node cannot simultaneously be in both these states.

INVARIANT 2. *For every node $y \in V$, either $M_{up}(y) = \emptyset$ or $M_{down}(y) = \emptyset$.*

Invariant 3 states that if a node y has up-marked or down-marked an incident edge (x, y), then the shadow-level $\ell_x(x, y)$ of the other endpoint x is no more than the level of y. Intuitively, the node y up-marks or down-marks an incident edge only if it wants to change its weight W_y without changing its own level $\ell(y)$. Suppose that the invariant is false, i.e., the node y has up-marked or down-marked an edge (x, y) with $\ell_x(x, y) > \ell(y)$. Then we have $\ell_y(x, y) \leq \ell(y) + 1 \leq \ell_x(x, y)$, where the first inequality follows from Invariant 1. But, this implies that y can *never change the weight $w(x, y)$ by up-marking or down-marking* (x, y), for the value of $w(x, y)$ is determined by the shadow-level of the other endpoint x. Thus, the node y does not gain anything by up-marking or down-marking the edge (x, y). This is why we guarantee the following invariant.

INVARIANT 3. *For every edge $(x, y) \in E$, if $\ell_y(x, y) \neq \ell(y)$, then we must have $\ell_x(x, y) \leq \ell(y)$.*

Six different states. For technical reasons, we will require that a node is always in one of six possible *states*. See Table 1. It is easy to check that this is sufficient to ensure conditions (2), (3) of Definition 3.1. See Lemma 3.2. One way to classify these states is as follows. Definition 3.1 requires that the weight of a node y lies in the range $0 \leq W_y < 1$. We partition this range into four intervals: I_1, I_2, I_3 and I_4. These intervals are non-empty as long as β is a sufficiently large constant.

$$I_1 = [0, f(\beta)), \quad I_2 = [f(\beta), 1 - 2/\beta)$$
$$I_3 = [1 - 2/\beta, 1 - 1/\beta) \text{ and } I_4 = [1 - 1/\beta, 1).$$

A node y is in UP state when $W_y \in I_4$, DOWN state when $W_y \in I_2$, and SLACK state when $W_y \in I_1$. As per Table 1, the node y has to satisfy some additional constraints when $\text{STATE}[y] \in \{\text{UP}, \text{DOWN}, \text{SLACK}\}$. Finally, if $W_y \in I_3$, then y is in one of three possible states – IDLE, UP-B, DOWN-B – depending on whether or not it has up-marked or down-marked any incident edge. By Invariant 2, a node cannot simultaneously up-mark some incident edges and down-mark some other incident edges. Hence, three cases can occur when $W_y \in I_3$. (a) $M_{up}(y) = M_{down}(y) = \emptyset$. In this case y is in IDLE state. (b) $M_{down}(y) = \emptyset$ and $M_{up}(y) \neq \emptyset$. In this case y is in UP-B state. (c) $M_{up}(y) = \emptyset$ and $M_{down}(y) \neq \emptyset$. In this case y is in DOWN-B state.

The six states are precisely defined in Table 1.

LEMMA 3.2. *If a node $y \in V$ is in one of the states described in Table 1, then its weight W_y satisfies conditions (2) and (3) of Definition 3.1.*

Proof. In every state, we have $0 \leq W_y < 1$ (see Table 1). We consider two mutually exclusive and exhaustive cases. (a) $f(\beta) \leq W_y < 1$. (b) $0 \leq W < f(\beta)$. In case (a), clearly the node-weight W_y satisfies conditions (2), (3) of Definition 3.1. In case (b), the node must be in SLACK state (see Table 1), and so we must have $\ell(y) = K$. Thus, the node satisfies conditions (2), (3) of Definition 3.1 even in case (b).

Note that each of the intervals I_1, I_2, I_3 and I_4 defined above is of length at least $1/\beta$ (see equation 3.5). On the other hand, for every edge $(u, v) \in E$ we have $w(u, v) \leq 1/\beta^K$, for K is the minimum possible level in a nice-partition. Accordingly, a natural or induced activation of a node can change its weight by at most $1/\beta^K$. Note that $1/\beta^K$ is much smaller than $1/\beta$. This apparently simple observation has an important implication, namely, that a node must be activated at least β^{K-1} times for its weight to cross the feasible range of any interval in $\{I_1, I_2, I_3, I_4\}$. As a corollary, if a node y has, say, $W_y \in I_3$ just before getting activated, then the activation can only move W_y to a *neighbouring* interval – I_2 or I_4. But it is not possible to have $W_y \in I_3$ just before the activation, and $W_y \in I_1$ just after the activation. Throughout the rest of the paper, we will be using this observation each time we consider the effect of an activation on a node. Next, we will briefly explain the motivation behind considering all these different states.

1. $\text{STATE}[y] = \text{UP}$. See row (1) in Table 1.
A node y is in UP state when $1 - 1/\beta \leq W_y < 1$. In this state the node's weight is close to one. Hence, whenever its weight increases further due to an activation the node tries to up-mark some incident edges from $E_{\ell(y)}(y)$, in the hope that this would reduce the node's weight and ensure that W_y never exceeds one. The node y can up-mark an edge only if the set $E_{\ell(y)}(y)$ is nonempty. Hence, we require that $E_{\ell(y)}(y) \neq \emptyset$. Further, to ensure that a up-marking does not violate Invariant 2, we require that $M_{down}(y) = \emptyset$.

2. $\text{STATE}[y] = \text{DOWN}$. See row (2) in Table 1.
A node y is in DOWN state when $f(\beta) \leq W_y < 1 - 2/\beta$. In this state the node's weight is close to the threshold

STATE[y]	Weight-range	Up-marked edges	Down-marked edges	Other constraints
1. UP	$1 - \frac{1}{\beta} \leq W_y < 1$		$M_{down}(y) = \emptyset$	$E_{\ell(y)}(y) \neq \emptyset$
2. DOWN	$f(\beta) \leq W_y < 1 - \frac{2}{\beta}$	$M_{up}(y) = \emptyset$		If $\ell(y) > K$, then $E_{\ell(y)}(y) - M_{down}(y) \neq \emptyset$
3. SLACK	$0 \leq W_y < f(\beta)$	$M_{up}(y) = \emptyset$	$M_{down}(y) = \emptyset$	$\ell(y) = K$
4. IDLE	$1 - \frac{2}{\beta} \leq W_y < 1 - \frac{1}{\beta}$	$M_{up}(y) = \emptyset$	$M_{down}(y) = \emptyset$	
5. UP-B	$1 - \frac{2}{\beta} \leq W_y < 1 - \frac{1}{\beta}$	$M_{up}(y) \neq \emptyset$	$M_{down}(y) = \emptyset$	
6. DOWN-B	$1 - \frac{2}{\beta} \leq W_y < 1 - \frac{1}{\beta}$	$M_{up}(y) = \emptyset$	$M_{down}(y) \neq \emptyset$	

Table 1: Constraints satisfied by a node in different states.

$f(\beta)$. There are two cases to consider here, depending on the current level of the node.

2-a. $\ell(y) > K$. In this case, whenever the value of W_y decreases further due to an activation, the node tries to down-mark some incident edges from $E_{\ell(y)}(y) \setminus M_{down}(y)$, in the hope that this would increase the node's weight and ensure that W_y does not drop below the threshold $f(\beta)$. The node y can down-mark an edge only if the set $E_{\ell(y)}(y) \setminus M_{down}(y)$ is nonempty. Hence, we require that $E_{\ell(y)}(y) \setminus M_{down}(y) \neq \emptyset$. Furthermore, in order to ensure that a down-marking does not violate Invariant 2, we require that $M_{up}(y) = \emptyset$.

2-b. $\ell(y) = K$. In this case, the node y cannot down-mark any incident edge (x, y), for we must always have $\ell_y(x, y) \in [K, L]$. Thus, we get $M_{down}(y) = \emptyset$ in addition to the constraints specified in row (2) of Table 1. If an activation makes W_y smaller than $f(\beta)$, then we simply set STATE$[y] \leftarrow$ SLACK.

We highlight one apparent discrepancy between the states UP and DOWN. If a node y is in DOWN state with $\ell(y) > K$, then it tries to down-mark some edges from $E_{\ell(y)}(y) \setminus M_{down}(y)$ after an activation that reduces its weight. However, if the same node is in UP state, then it tries to up-mark some edges from $E_{\ell(y)}(y)$ after an activation that increases its weight. This discrepancy is due to the fact that $E_{\ell(y)}(y) \cap M_{up}(y) = \emptyset$, as every edge $(x, y) \in M_{up}(y)$ has $\ell(x, y) \geq \ell_y(x, y) = \ell(y) + 1$. In other words, an edge up-marked by y can never belong to the set $E_{\ell(y)}(y)$, and hence $E_{\ell(y)}(y) \setminus M_{up}(y) = E_{\ell(y)}(y)$. In contrast, an edge $(x, y) \in M_{down}(y)$ belongs to the set $E_{\ell(y)}(x, y)$ if $\ell_x(x, y) = \ell(y)$.

3. STATE$[y]$ = SLACK. See row (3) in Table 1.
A node y is in SLACK state when $0 \leq W_y < f(\beta)$. In order to ensure condition (2) of Definition 3.1, we require that the node be at level K. Since K is the minimum possible level, there is no need for the node to prepare for moving down to a lower level in future. Hence, we require that $M_{down}(y) = \emptyset$. Further, the node's weight is currently so small that it will take quite some time before the node has to prepare for moving up to a higher level. Hence, we require that $M_{up}(y) = \emptyset$.

4. STATE$[y]$ = IDLE. See row (4) in Table 1.
A node y is in IDLE state when $1 - 2/\beta \leq W_y < 1 - 1/\beta$ and $M_{up}(y) = M_{down}(y) = \emptyset$. In this state the node's weight is neither too large nor too small, and the node does not have any up-marked or down-marked incident edges. Intuitively, the node need not worry even if its weight changes due to an activation in this state. In other words, when a node gets activated in IDLE state, it does not up-mark, down-mark or un-mark any of its incident edges. After a sufficiently large number of activations when the node's weight drops below (resp. rises above) the threshold $1 - 2/\beta$ (resp. $1 - 1/\beta$), it switches to the state DOWN (resp. UP).

5. STATE$[y]$ = UP-B. See row (5) in Table 1. The term "UP-B" stands for "UP-BACKTRACK".
A node y is in UP-B state when $1 - 2/\beta \leq W_y < 1 - 1/\beta$, $M_{up}(y) \neq \emptyset$ and $M_{down}(y) = \emptyset$. Intuitively, this state of the node captures the following scenario. Some time back the node y was in UP state with $1 - 1/\beta \leq W_y < 1$, $M_{up}(y) \neq \emptyset$ and $M_{down}(y) = \emptyset$. From that point onward, the node encountered a large number of activations that kept on reducing its weight. Eventually, the value of W_y became smaller than $1 - 1/\beta$ and the node entered the state UP-B. If the node keeps getting activated in this manner, then in near future W_y will become smaller than $1 - 2/\beta$ and the node y will have to enter the state DOWN. At that time we must have $M_{up}(y) = \emptyset$. In other words, the node y has to ensure that $M_{up}(y) = \emptyset$ before its weight drops below the threshold $1 - 2/\beta$. Thus, whenever STATE$[y]$ = UP-B and the node-weight W_y decreases due to an activation, the node y un-marks some edges from $M_{up}(y)$.

6. STATE$[y]$ = DOWN-B. See row (6) in Table 1. The term "DOWN-B" stands for "DOWN-BACKTRACK".
A node y is in DOWN-B state when $1 - 2/\beta \leq W_y < 1 - 1/\beta$, $M_{down}(y) \neq \emptyset$ and $M_{up}(y) = \emptyset$. Intuitively, this state of the node captures the following scenario. Some time back the node y was in DOWN state with $f(\beta) \leq W_y < 1 - 2/\beta$, $M_{down}(y) \neq \emptyset$ and $M_{up}(y) = \emptyset$. From that point onward, the node encountered a large number of activations that kept on increasing its weight. Eventually, the value of W_y became greater than $1 - 2/\beta$ and the node entered the state DOWN-B. If the node keeps getting activated in this manner, then in near future W_y will become greater than $1 - 1/\beta$ and it will have to enter the state UP. At that time we must have $M_{down}(y) = \emptyset$. In other words, the node y has to ensure that $M_{down}(y) = \emptyset$ before its weight increases beyond the threshold $1 - 1/\beta$. Thus, whenever STATE$[y]$ = DOWN-B and W_y increases due to an activation, the node y un-marks some edges from $M_{down}(y)$.

3.3 Dirty nodes. Our algorithm maintains a bit $D[y] \in \{0, 1\}$ associated with each node $y \in V$. We say that the node y is *dirty* if $D[y] = 1$ and *clean* otherwise. Intuitively, the node y is dirty when it is unsatisfied about its current condition and it wants to up-mark, down-mark or un-mark some of its incident edges. Once a dirty node is done with up-marking, down-marking or un-marking the relevant edges, it becomes clean again.

In our algorithm, a node becomes dirty only after it encounters a natural or induced activation. The converse of this statement, however, is not true. There may be times when a node remains clean even after

getting activated, and this will be crucial in bounding the worst-case update time of our algorithm. Whether or not a node will become dirty due to an activation depends on: (1) the state of the node, (2) the type of the activation under consideration (whether it increases or decreases the node-weight), and (3) the node's current level. We have three rules that determine when a node becomes dirty.

RULE 3.1. *A node y with* STATE$[y] \in \{$UP, DOWN-B$\}$ *becomes dirty after an activation that increases its weight. In contrast, such a node* does not *become dirty after an activation that decreases its weight.*

Justification for Rule 3.1.

Case 1. STATE$[y]$ = UP. Here, we have $1 - 1/\beta \leq W_y < 1$ and $M_{down}(y) = \emptyset$. If an activation increases the value of W_y, then y needs to up-mark some edges from $E_{\ell(y)}(y)$, in the hope that W_y remains smaller than 1 (see the discussion in Section 3.2). Hence, the node becomes dirty. In contrast, if an activation reduces the value of W_y, then y need not up-mark, down-mark or un-mark any of its incident edges. Due to this inaction, if it so happens that $1 - 2/\beta \leq W_y < 1 - 1/\beta$ after the activation, then the node simply switches to state UP-B or IDLE depending on whether or not $M_{up}(y) \neq \emptyset$.

Case 2. STATE$[y]$ = DOWN-B. Here, we have $1 - 2/\beta \leq W_y < 1 - 1/\beta$, $M_{down}(y) \neq \emptyset$ and $M_{up}(y) = \emptyset$. Such a node must un-mark all its incident edges before its weight rises past the threshold $1 - 1/\beta$ (see the discussion in Section 3.2). Hence, whenever its weight increases due to an activation and STATE$[y]$ = DOWN-B, the node y becomes dirty and un-marks some edges from $M_{down}(y)$. In contrast, if an activation reduces its weight, then the node y need not up-mark, down-mark or un-mark any of its incident edges. Due to this inaction, if it so happens that $f(\beta) \leq W_y < 1 - 2/\beta$ after the activation, then we set STATE$[y] \leftarrow$ DOWN. At this point, if we have $E_{\ell(y)}(y) \setminus M_{down}(y) = \emptyset$ and $\ell(y) > K$, then the node y moves to a lower level while being in DOWN state (see Case 2-b in Section 4.1).

RULE 3.2. *Consider a node y such that either (1)* STATE$[y]$ = DOWN *and $\ell(y) > K$, or (2)* STATE$[y]$ = UP-B. *This node becomes dirty after an activation that decreases its weight. In contrast, the node does not become dirty after an activation that increases its weight.*

Justification for Rule 3.2.

Case 1. STATE$[y]$ = DOWN and $\ell(y) > K$. Thus, we have $f(\beta) \leq W_y < 1 - 2/\beta$ and $M_{up}(y) = \emptyset$. If an activation decreases its weight, then y needs to down-mark some edges from $E_{\ell(y)}(y) \setminus M_{down}(y)$, in the hope that W_y does not become smaller than $f(\beta)$ (see the discussion in Section 3.2). Hence, the node y becomes dirty. In contrast, if an activation increases its weight, then y need not up-mark, down-mark or un-mark any of its incident edges. Due to this inaction, if it so happens that $1 - 2/\beta \leq W_y < 1 - 1/\beta$ after the activation, then the node simply switches to state DOWN-B or IDLE depending on whether or not $M_{down}(y) \neq \emptyset$.

Case 2. STATE$[y]$ = UP-B. Thus, we have $1 - 2/\beta \leq W_y < 1 - 1/\beta$, $M_{up}(y) \neq \emptyset$ and $M_{down}(y) = \emptyset$. Such a node must un-mark all its incident edges before its weight drops below the threshold $1 - 2/\beta$ (see the discussion in Section 3.2). Hence, whenever its weight decreases due to an activation and STATE$[y]$ = UP-B, the node y becomes dirty and un-marks some edges from $M_{up}(y)$. In contrast, if an activation increases its weight, then y need not up-mark, down-mark or un-mark any of its incident edges. Due to this inaction, if it so happens that $1 - 1/\beta \leq W_y < 1$ after the activation, then we set STATE$[y] \leftarrow$ UP. At this point, if we have $E_{\ell(y)}(y) = \emptyset$, then the node y moves to a higher level while being in UP state (see Case 2-a in Section 4.1).

RULE 3.3. *A node y with either (1)* STATE$[y] \in \{$SLACK, IDLE$\}$ *or (2)* $\{$STATE$[y]$ = DOWN *and $\ell(y) = K\}$ never becomes dirty after an activation.*

Justification for Rule 3.3.

Case 1. STATE$[y]$ = SLACK. Here, we have $0 \leq W_y < f(\beta)$, $M_{up}(y) = M_{down}(y) = \emptyset$ and $\ell(y) = K$. When such a node gets activated, it need not up-mark or down-mark any of its incident edges. Due to this inaction, if it so happens that $f(\beta) \leq W_y < 1 - 2/\beta$ after the activation, then we set STATE$[y] \leftarrow$ DOWN.

Case 2. STATE$[y]$ = IDLE. Here, we have $1 - 2/\beta \leq W_y < 1 - 1/\beta$ and $M_{up}(y) = M_{down}(y) = \emptyset$. When such a node gets activated, it need not up-mark or down-mark any of its incident edges. Due to this inaction, if it so happens that $1 - 1/\beta \leq W_y < 1$ after the activation, then we set STATE$[y] \leftarrow$ UP. At this point, if we have $E_{\ell(y)}(y) = \emptyset$, then the node y moves to a higher level while being in UP state (see Case 2-a in Section 4.1). In contrast, if it so happens that $f(\beta) \leq W_y < 1 - 2/\beta$ after the activation, then we set STATE$[y] \leftarrow$ DOWN. At this point, if we have $E_{\ell(y)}(y) \setminus M_{down}(y) = \emptyset$ and $\ell(y) > K$, then the node y moves to a lower level while being in DOWN state (see Case 2-b in Section 4.1).

Case 3. STATE$[y]$ = DOWN and $\ell(y) = K$. Thus, we have $f(\beta) \leq W_y < 1 - 2/\beta$ and $M_{up}(y) = \emptyset$. Since $\ell(y) = K$ and $\ell_y(x, y) \in [K, L]$ for every edge $(x, y) \in E$, we also have $M_{down}(y) = \emptyset$. When such a node gets activated, it need not up-mark or down-mark any of its incident edges. Due to this inaction, if we have $0 \leq W_y < f(\beta)$ after the activation, then we set STATE$[y] \leftarrow$

SLACK. In contrast, if $1 - 2/\beta \leq W_y < 1 - 1/\beta$ after the activation, then we set STATE$[y] \leftarrow$ IDLE.

COROLLARY 3.2. *If an activation of a node y makes it dirty, then the state of the node remains the same just before and just after the activation (see Section 6.1).*

Proof. While justifying Rules 3.1 – 3.3, whenever we changed the state of the node y due to an activation, we ensured that the node did not become dirty.

3.4 Data structures. In our dynamic algorithm, every node $y \in V$ maintains the following data structures.

1. Its weight W_y, level $\ell(y)$, and state STATE$[y] \in$ {UP, DOWN, SLACK, IDLE, UP-B, DOWN-B}.
2. The sets $M_{up}(y), M_{down}(y)$ as balanced search trees.
3. A bit $D[y] \in \{0, 1\}$ to indicate if the node y is *dirty*.
4. For every level $i \in \{0, \ldots, L\}$, the set of edges $E_i(y)$ as a balanced search tree.

Furthermore, every edge $(x, y) \in E$ maintains the values of its weight $w(x, y)$ and level $\ell(x, y)$.

Remark about maintaining the shadow-levels. Note that we do not *explicitly* maintain the shadow-level $\ell_x(x, y)$ of a node $y \in V$ with respect to an edge $(x, y) \in E$. This is due to the following reason.

For the sake of contradiction, suppose that our algorithm in fact maintains the values of the shadow-levels $\ell_y(x, y)$. Consider a scenario where the node y has STATE$[y]$ = UP, $\ell(y) = i$, and the value of W_y is very close to one. Next, suppose that an activation increases the value of W_y, and the node up-marks one or more edges from the set $E_i(y)$ to ensure that the value of W_y remains smaller than one. Since $\ell(x, y) = i + 1$ for every edge $(x, y) \in M_{up}(y)$, all the newly up-marked edges get deleted from the set $E_i(y)$ and added to the set $E_{i+1}(y)$. At this point, we might end up in a situation where $E_i(y) = \emptyset$, which violates a constraint of row (1) in Table 1. Our algorithm deals with this issue by moving the node y up to level $(i + 1)$, i.e., by setting $\ell(y) \leftarrow (i + 1)$. Since $E_i(y) = \emptyset$, this does not affect the weight of any edge. However, for every edge $(x, y) \in E$ with $\ell(x, y) > i+1$, the shadow-level $\ell_y(x, y)$ changes from i to $(i + 1)$. Since each edge $(x, y) \in E$ with $\ell(x, y) > i + 1$ has weight at most $\beta^{-(i+2)}$, and since $W_y < 1$, there can be $\beta^{i+2} - 1$ many such edges. Accordingly, the node y might be forced to change the values of the shadow-levels $\ell_y(x, y)$ for $O(\beta^{i+2})$ many edges (x, y). The worst-case update time then becomes $O(\beta^{i+2})$, which is polynomial in n for large values of i.

We avoid this problem by giving up on explicitly maintaining the values of the shadow-levels $\ell_y(x, y)$. Still we can determine the value of $\ell_x(x, y)$ in $O(\log n)$ time from the data structures that *are* in fact maintained by us. Specifically, we know that if $(x, y) \in M_{up}(y)$, then $\ell_y(x, y) = \ell(y) + 1$. Else if $(x, y) \in M_{down}(y)$, then $\ell_y(x, y) = \ell(y) - 1$. Finally, else if $(u, y) \notin M_{up}(y) \cup M_{down}(y)$, then $\ell_y(x, y) = \ell(y)$.

For ease of exposition, we nevertheless use the notation $\ell_y(x, y)$ while describing our algorithm in subsequent sections. Whenever we do this, the reader should keep it in mind that we are implicitly computing $\ell_y(x, y)$ as per the above procedure.

4 Some basic subroutines

4.1 The subroutine UPDATE-STATUS(y). This subroutine is called each time a node y experiences a natural or an induced activation. This tries to ensure, by changing the state and level of y if necessary, that y satisfies the constraints specified in Table 1. If the subroutine fails to ensure this condition, then our algorithm HALTS. During the analysis of our algorithm, we will prove that it never HALTS due to a call to UPDATE-STATUS(y). This implies that every node satisfies the constraints in Table 1, and hence Lemma 3.2 guarantees that conditions (2) and (3) of Definition 3.1 continue to remain satisfied all the time.

We say that a node is *fit* in a state $X \in$ {UP, DOWN, SLACK, IDLE, UP-B, DOWN-B} if it satisfies all the constraints for state X as specified in Table 1, and *unfit* otherwise. If $D[y] = 1$, then our algorithm HALTS if y is unfit in its *current* state. In contrast, if $D[y] = 0$, then our algorithm HALTS if y is unfit in *every* state, albeit with one caveat: If the node is unfit in either state UP or state DOWN, then we first try to make it fit in that state by changing its level $\ell(y)$. Hence, there is a sharp distinction between the treatments received by the clean nodes on the one hand and the dirty nodes on the other. Specifically, the state of a node y can change during to a call to UPDATE-STATUS(y) only if y is clean at the beginning of the call. This distinction comes from Corollary 3.2, which requires that a node does not change its state if it becomes dirty. We now describe the subroutine in details.

Case 1. $D[y] = 1$. The node y is dirty.
If y is fit in its current state, then we terminate the subroutine. Otherwise our algorithm HALTS.

Case 2. $D[y] = 0$. The node y is clean.
If we can find some state $X \in$ {UP, DOWN, SLACK, IDLE, UP-B, DOWN-B} in which y is fit, then we set STATE$[y] \leftarrow X$ and terminate the subroutine. Else if the node y is unfit in every state, then we consider the sub-cases 2-a, 2-b and 2-c.

Case 2-a. The node is unfit in state UP only due to the last constraint in row (1) of Table 1. Thus, we have

$1-1/\beta \leq W_y < 1$, $M_{down}(y) = \emptyset$ and $E_{\ell(y)}(y) = \emptyset$. Let $i \leftarrow \ell(y)$ be the current level of y. We find the minimum level $j > i$ where $E_j(y) \neq \emptyset$. Such a level j must exist since $W_y > 0$. We move the node y up to level j by setting $\ell(y) \leftarrow j$. This does not change the weight of any edge. Furthermore, when the node was in level i, we had $\ell_y(x,y) \leq \ell(y) + 1 \leq i + 1 \leq j$ for every edge $(x,y) \in E$ incident on y (see Invariant 1). Hence, after the node moves up to level j, we have $\ell_y(x,y) = j = \ell(y)$ for every edge $(x,y) \in E$. In other words, the node y is not supposed to have any up-marked edges incident on it just after moving to level j. Accordingly, we set $M_{up}(y) \leftarrow \emptyset$. Then we terminate the subroutine.

Case 2-b. The node is unfit in state DOWN only due to the last constraint in row (2) of Table 1. Thus, we have $f(\beta) \leq W_y < 1 - 2/\beta$, $M_{up}(y) = \emptyset$, $E_{\ell(y)}(y) \setminus M_{down}(y) = \emptyset$ and $\ell(y) > K$. Let $i \leftarrow \ell(y)$ be the current level of y. We first move the node down to level $i-1$ by setting $\ell(y) \leftarrow i-1$. We claim that this does not change the level (and weight) of any edge. To see why the claim is true, consider any edge $(x,y) \in E$ incident on y. Since $M_{up}(y) = \emptyset$, we must have $\ell_y(x,y) \leq i$ just before the node moves down to level $(i-1)$. If $\ell_x(x,y) \geq i$, then the value of $\ell(x,y)$ is determined by the other endpoint x and the level of such an edge does not change as y moves down to level $(i-1)$. In contrast, if $\ell_x(x,y) < i$, then we have $(x,y) \in M_{down}(y)$: for otherwise the edge (x,y) will belong to the set $E_i(y) \setminus M_{down}(y)$ which we have assumed to be empty. The level of such an edge remains equal to $(i-1)$ as the node y moves down from level i to level $(i-1)$. This concludes the proof of the claim that the edge-weights do not change as y moves down from level i to level $(i-1)$. Next, consider any edge (x,y) that was down-marked when the node y was at level i. At that time, we had $\ell_y(x,y) = i-1$. Hence, after the node moves down to level $i-1$, we get $\ell_y(x,y) = i-1 = \ell(y)$. Thus, the node cannot have any down-marked edge incident on it just after moving down to level $i-1$. Accordingly, we set $M_{down}(y) \leftarrow \emptyset$. At this point, if we find that $E_{i-1}(y) = \emptyset$, then we move the node further down to the lowest level K, by setting $\ell(y) \leftarrow K$. This does not change the level and weight of any edge in the graph. Finally, we terminate the subroutine.

Case 2-c. In every scenario other than 2-a and 2-b described above, our algorithm HALTS.

A note on the space complexity. In cases 2-a and 2-b of the above procedure, there is a step where we set $M_{up}(y) \leftarrow \emptyset$ and $M_{down}(y) \leftarrow \emptyset$ respectively. It is essential to execute this step in $O(\text{poly}\log n)$ time: otherwise we cannot claim that the update time of our algorithm is $O(\text{poly}\log n)$ in the worst-case. Unfortunately for us, there can be $\Omega(\beta^{\ell(y)})$ many edges in the set $M_{up}(y)$ or $M_{down}(y)$. Hence, it will take $\Omega(\beta^{\ell(y)})$ time to empty that set if we have to delete all those edges from the corresponding balanced search tree. Note that $\beta^{\ell(y)} = \Omega(n)$ for large $\ell(y)$.

To address this concern, we maintain two pointers $root[M_{up}(y)]$ and $root[M_{down}(y)]$ for each node $y \in V$. They respectively point to the root of the balanced search tree for $M_{up}(y)$ and $M_{down}(y)$. When we want to set $M_{up}(y) \leftarrow \emptyset$ or $M_{down}(y) \leftarrow \emptyset$, we respectively set $root[M_{up}(y)] \leftarrow$ NULL or $root[M_{down}(y)] \leftarrow$ NULL. This takes only constant time. The downside of this approach is that the algorithm now uses up a lot of *junk space* in memory: This space is occupied by the balanced search trees that were *emptied* in the past. As a result, the space complexity of the algorithm becomes $O(t \text{ poly}\log n)$ for handling a sequence of t edge insertions/deletions starting from an empty graph. This is due to the fact that our algorithm will be shown to have a worst-case update time of $O(\text{poly}\log n)$. Hence, we can upper bound the total time taken to handle these edge insertions/deletions by $O(t \text{ poly}\log n)$, and this, in turn, gives a trivial upper bound on the amount of *junk space* used up in the memory.

A standard way to bring down the space complexity is to run a *clean-up* algorithm *in the background*. Each time an edge is inserted into or deleted from the graph, we visit $O(\text{poly}\log n)$ memory cells that are currently junk and *free them up*. Thus, the worst case update time of the clean-up algorithm is also $O(\text{poly}\log n)$, and this increases the overall update time of our scheme by only a $O(\text{poly}\log n)$ factor. The size of all sets $M_{up}(.)$ and $M_{down}(.)$ that exist at a given point in time is $O(m)$. Hence, this clean-up algorithm is at most $O(m)$ space "behind", i.e., the additional space requirement for junk space is $O(m)$. For ease of exposition, from this point onward we will simply assume that we can empty a balanced search tree in $O(1)$ time.

LEMMA 4.1. *The subroutine* UPDATE-STATUS(y) *takes* $O(\log n)$ *time.*

Proof. Case 1 can clearly be implemented in $O(1)$ time. In case 2-a, we have to find the minimum level $j > i$ where $E_j(y) \neq \emptyset$. This operation takes time proportional to the number of levels, which is $L - K + 1 = O(\log n)$. Everything else takes $O(1)$ time. Finally, case 2-b and case 2-c also take $O(1)$ time.

4.2 The subroutine PIVOT-UP$(v,(u,v))$. This is described in Figure 3. This subroutine is called when the node v is dirty and it wants to increase its shadow-level $\ell_v(u,v)$ with respect to the edge (u,v). There are two situations under which such an event can take place: (1) STATE$[v]$ = UP and v wants to up-mark the

edge (u, v), and (2) STATE$[v]$ = DOWN-B and v wants to un-mark the edge (u, v). The subroutine PIVOT-UP$(v, (u, v))$ updates the relevant data structures, decides whether the node u should become dirty because of this event, and returns TRUE if the event changes the weight of the edge (u, v) and FALSE otherwise. Thus, if the subroutine returns TRUE, then this amounts to an induced activation of the node u.

The subroutine MOVE-UP$(v, (u, v))$. Step (01) in Figure 3 calls another subroutine MOVE-UP$(v, (u, v))$. This subroutine updates the relevant data structures as the value of $\ell_v(u, v)$ increases by one, and returns TRUE if the weight $w(u, v)$ gets changed and FALSE otherwise. To see an example where MOVE-UP$(v, (u, v))$ returns FALSE, consider a situation where STATE$[v]$ = DOWN-B, $\ell(v) = i$, $\ell_v(u, v) = i - 1$, and $\ell_u(u, v) = \ell(u) = i$. In this instance, even after the node v increases the value of $\ell_v(u, v)$ by un-marking the edge (u, v), the weight $w(u, v)$ does not change.

The subroutine MOVE-UP$(v, (u, v))$ ensures that Invariant 3 remains satisfied. Specifically, after the value of $\ell_v(u, v)$ increases we might have $\ell_v(u, v) > \ell(u)$, and then we must ensure that the edge $(u, v) \notin M_{up}(u) \cup M_{down}(u)$: otherwise Invariant 3 will be violated (set $y = u$ and $x = v$ in Invariant 3). If we end up in this situation, then the subroutine MOVE-UP$(v, (u, v))$ removes the edge from $M_{up}(u) \cup M_{down}(u)$.

01. $Y \leftarrow$ MOVE-UP$(v, (u, v))$
02. IF Y = TRUE and $\{$either STATE$[u]$ = UP-B or (STATE$[u]$ = DOWN and $\ell(u) > K$)$\}$
03. $D[u] \leftarrow 1$
04. UPDATE-STATUS(u)
05. RETURN Y.

Figure 3: PIVOT-UP$(v, (u, v))$.

Deciding if the node u becomes dirty. We now continue with the description of the subroutine PIVOT-UP$(v, (u, v))$. After step (01) in Figure 3, it remains to decide whether the node u should become dirty. This decision is made following the three rules specified in Section 3.3. Note that if we increase the value of $\ell_v(u, v)$, then it can never lead to an increase in the weight W_u. Thus, if Y = TRUE, then it means that the weight W_u dropped during the call to the subroutine MOVE-UP$(v, (u, v))$. On the other hand, if Y = FALSE, then it means that the weight W_u did not change during the call to the subroutine MOVE-UP$(v, (u, v))$. In this event, the node u never becomes dirty.

As per Rules 3.1 – 3.3, if the weight W_u gets reduced, then u becomes dirty iff either STATE$[u]$ = UP-B or (STATE$[u]$ = DOWN, $\ell(u) > K$). Thus, the subroutine sets $D[u] \leftarrow 1$ iff two conditions are satisfied: (1) Y = TRUE, and (2) either STATE$[u]$ = UP-B or (STATE$[u]$ = DOWN, $\ell(u) > K$).

Finally, just before terminating the subroutine PIVOT-UP$(v, (u, v))$ in Figure 3, we call the subroutine UPDATE-STATUS(u). The reason for this call is explained in the beginning of Section 4.1.

LEMMA 4.2. *The subroutine PIVOT-UP$(v, (u, v))$ takes $O(\log n)$ time. It returns* TRUE *if the weight $w(u, v)$ gets changed, and* FALSE *otherwise. The node u becomes dirty only if the subroutine returns* TRUE.

Proof. A call to the subroutine UPDATE-STATUS(y) takes $O(\log n)$ time, as per Lemma 4.1. The rest of the proof follows from the description of the subroutine.

4.3 The subroutine PIVOT-DOWN$(v, (u, v))$. This is described in Figure 4. This subroutine is called when the node v is dirty and it wants to decrease its shadow-level $\ell_v(u, v)$ with respect to the edge (u, v). There are two situations under which such an event can take place: (1) STATE$[v]$ = DOWN and v wants to down-mark the edge (u, v), and (2) STATE$[v]$ = UP-B and v wants to un-mark the edge (u, v). The subroutine PIVOT-DOWN$(v, (u, v))$ updates the relevant data structures, decides whether the node u should become dirty, and returns TRUE if the weight of the edge (u, v) gets changed and FALSE otherwise. Thus, if the subroutine returns TRUE, then this amounts to an induced activation of the node u. This subroutine, however, is *not* a mirror-image of the subroutine PIVOT-UP$(v, (u, v))$. The difference between them is explained below.

In the subroutine PIVOT-DOWN$(v, (u, v))$, suppose that the node v has decreased the value of $\ell_v(u, v)$, and this has increased the weight W_u. Furthermore, the node u is currently in a state where Rules 3.1 – 3.3 dictate that it should become dirty when its weight increases. If this is the case, then the node u attempts to *undo* its weight-change by increasing the value of $\ell_u(u, v)$. To take a concrete example, suppose that just before the subroutine PIVOT-DOWN$(v, (u, v))$ is called, we have STATE$[v]$ = UP-B, $\ell(v) = i$, $\ell_v(u, v) = i + 1$, STATE$[u]$ = UP, $\ell(u) = i$ and $\ell_u(u, v) = i$. The node v now decreases the value of $\ell_v(u, v)$ by one, and un-marks the edge (u, v). Thus, the weight $w(u, v)$ changes from $\beta^{-(i+1)}$ to β^{-i}. This also increases the weight W_u by an amount $\beta^{-i} - \beta^{-(i+1)}$. The node u will now undo this change by up-marking the edge (u, v), which will increase $\ell_u(u, v)$ by one. This will bring the weight W_u back to its initial value. In contrast, the subroutine PIVOT-UP$(v, (u, v))$ does not allow the node u to perform such "undo" operations. This "undo" operation performed by u in PIVOT-DOWN$(v, (u, v))$ will be

crucial in bounding the update time of our algorithm.

The subroutine MOVE-DOWN$(v,(u,v))$. Step (01) in Figure 4 calls another subroutine MOVE-DOWN$(v,(u,v))$. This subroutine updates the relevant data structures as the value of $\ell_v(u,v)$ decreases by one, and returns TRUE if the weight $w(u,v)$ gets changed and FALSE otherwise. To see an example where MOVE-DOWN$(v,(u,v))$ returns FALSE, consider a situation where STATE$[v]$ = DOWN, $\ell(v) = i$, $\ell_v(u,v) = i$, and $\ell_u(u,v) = \ell(u) = i$. In this instance, even after the node v decreases the value of $\ell_v(u,v)$ by down-marking the edge (u,v), the weight $w(u,v)$ does not change.

```
01. Y ← MOVE-DOWN(v,(u,v))
02. IF STATE[u] = UP
03.     IF Y = TRUE
04.         IF (u,v) ∉ M_up(u) ⋀ ℓ(u) ≥ ℓ_v(u,v)
05.             MOVE-UP(u,(u,v))
06.             UPDATE-STATUS(u)
07.             RETURN FALSE
08.         ELSE
09.             D[u] ← 1
10.             UPDATE-STATUS(u)
11.             RETURN Y.
12.     ELSE
13.         UPDATE-STATUS(u)
14.         RETURN Y.
15. ELSE IF STATE[u] = DOWN-B
16.     IF Y = TRUE, THEN
17.         IF (u,v) ∈ M_down(u) ⋀ ℓ_v(u,v) < ℓ(u)
18.             MOVE-UP(u,(u,v))
19.             UPDATE-STATUS(u)
20.             RETURN FALSE
21.         ELSE
22.             D[u] ← 1
23.             UPDATE-STATUS(u)
24.             RETURN Y.
25.     ELSE
26.         UPDATE-STATUS(u)
27.         RETURN Y.
28. ELSE
29.     UPDATE-STATUS(u)
30.     RETURN Y.
```

Figure 4: PIVOT-DOWN$(v,(u,v))$. Steps (05) and (18) correspond to "undo" operations by the node u.

Unlike the subroutine MOVE-UP$(v,(u,v))$, here we need not worry about Invariant 3 getting violated, for the following reason. Set $v = x$ and $u = y$ in Invariant 3 just as we did while considering the subroutine MOVE-UP$(v,(u,v))$. If $\ell_u(u,v) = \ell(u)$, the Invariant 3 clearly remains satisfied even as $\ell_v(u,v)$ decreases by one. If $\ell_u(u,v) \neq \ell(u)$, then by Invariant 3 we have $\ell_v(u,v) \leq \ell(u)$ just before the call to MOVE-DOWN$(v,(u,v))$. In this case as well, Invariant 3 continues to remain satisfied even as $\ell_v(u,v)$ decreases by one.

Deciding if the node u becomes dirty. We continue with the description of PIVOT-DOWN$(v,(u,v))$. After step (01) in Figure 4, it remains to decide whether (a) the node u is about to become dirty, and if the answer is yes, then whether (b) the node can escape this fate by successfully executing an "undo" operation. Decision (a) is taken following the Rules 3.1 – 3.3.

Since the subroutine PIVOT-DOWN$(v,(u,v))$ is called when the node v wants to decrease the value of $\ell_v(u,v)$, this can never lead to a decrease in the value of W_u. In other words, step (01) in Figure 4 can only increase the weight W_u. Specifically, if Y = TRUE, then the weight W_y increases. In contrast, if Y = FALSE, then the weight W_u does not change at all. In the latter event, the node u never becomes dirty, and the question of u attempting to execute an "undo" operation does not arise. In the former event, Rules 3.1 – 3.3 dictate that the node u is about to become dirty iff STATE$[u] \in \{$UP, DOWN-B$\}$. This is the only situation where we have to check if the node u can execute a successful "undo" operation. This situation can be split into two mutually exclusive and exhaustive cases (1) and (2), as described below. In every other situation, the node u does not become dirty, it does not perform an undo operation, and the subroutine PIVOT-DOWN$(v,(u,v))$ returns the same value as Y. Finally, just before terminating the subroutine PIVOT-DOWN$(v,(u,v))$ we always call UPDATE-STATUS(u). The reason for this step is explained in the beginning of Section 4.1.

Case 1: Y = TRUE and STATE$[u]$ = UP.
See steps (03) – (11) in Figure 4. In this case, either $(u,v) \in M_{up}(u)$ or $(u,v) \notin M_{up}(u)$. In the former event, the edge (u,v) has already been up-marked by u, and hence u cannot increase the value of $\ell_u(u,v)$ any further. In the latter event, we have $\ell_u(u,v) = \ell(u)$. Before up-marking the edge (u,v), the node u should ensure that it satisfies Invariant 3 (set $u = y$ and $v = x$). Hence, we must have $\ell_v(u,v) \leq \ell(u)$ if the node u is to execute an undo operation. To summarise, we have to sub-cases.

Case 1-a: $(u,v) \notin M_{up}(u)$ and $\ell_v(u,v) \leq \ell(u)$. In this event, increasing the value of $\ell_u(u,v)$ by one changes the weight $w(u,v)$ from $\beta^{-\ell(u)}$ to $\beta^{-(\ell(u)+1)}$. This undo operation is performed by calling the subroutine MOVE-UP$(u,(u,v))$.

Case 1-b: Either $(u,v) \in M_{up}(u)$ and $\ell_v(u,v) > \ell(u)$. In this event, the node u cannot perform an undo operation and becomes dirty as per Rule 3.1.

Case 2: $Y = \text{TRUE}$ and $\text{STATE}[u] = \text{DOWN-B}$.
See steps (16) – (24) in Figure 4. In this case, either $(u, v) \in M_{down}(u)$ or $(u, v) \notin M_{down}(u)$. In the latter event, the only way u can increase the value of $\ell_u(u, v)$ is by up-marking the edge (u, v). But this would result in the set $M_{up}(u)$ becoming non-empty, which in turn would violate a constraint in row (6) of Table 1. Hence, the node u can perform an undo operation only if $(u, v) \in M_{down}(u)$. Further, if $\ell_v(u, v) \geq \ell(u)$ and $(u, v) \in M_{down}(u)$, then the weight $w(u, v)$ remains equal to $\beta^{-\ell_v(u,v)}$ even as the value of $\ell_u(u, v)$ changes from $\ell(u)-1$ to $\ell(u)$. This prevents u from executing an undo operation. To summarise, there are two sub-cases.

Case 2-a: We have $(u, v) \in M_{down}(u)$ and $\ell_v(u, v) < \ell(u)$. In this event, increasing the value of $\ell_u(u, v)$ by one changes the weight $w(u, v)$ from $\beta^{-(\ell(u)-1)}$ to $\beta^{-\ell(u)}$. This undo operation is performed by calling the subroutine MOVE-UP$(u, (u, v))$.

Case 2-b: Either $(u, v) \in M_{down}(u)$ or $\ell_v(u, v) \geq \ell(u)$. In this event, the node u cannot perform an undo operation and becomes dirty as per Rule 3.1.

LEMMA 4.3. *The subroutine PIVOT-DOWN$(v, (u, v))$ takes $O(\log n)$ time. It returns* TRUE *if the weight $w(u, v)$ gets changed, and* FALSE *otherwise. The node u becomes dirty only if the subroutine returns* TRUE.

Proof. A call to the subroutine UPDATE-STATUS(y) takes $O(\log n)$ time, as per Lemma 4.1. The rest of the proof follows from the description of the subroutine.

5 The subroutine FIX-DIRTY-NODE(v)

Note that the node v undergoes a natural activation when an edge (u, v) is inserted into or deleted from the graph. In contrast, the node v undergoes an induced activation when some neighbour x of v calls the subroutine PIVOT-UP$(x, (x, v))$ or PIVOT-DOWN$(x, (x, v))$, and that subroutine returns TRUE.

The subroutine FIX-DIRTY-NODE(v) is called immediately after the node v becomes dirty due to a natural or an induced activation. Depending on the current state of v, the subroutine up-marks, down-marks or unmarks some of its incident edges $(u, v) \in E$. This involves increasing or decreasing the shadow-level $\ell_v(u, v)$ by one, for which the subroutine respectively calls PIVOT-UP$(v, (u, v))$ or PIVOT-DOWN$(v, (u, v))$. We say that a given call to PIVOT-UP$(v, (u, v))$ or PIVOT-DOWN$(v, (u, v))$ is a *success* if the weight $w(u, v)$ gets changed due to the call (i.e., the call returns TRUE), and a *failure* otherwise (i.e., the call returns FALSE). We ensure that one call to the subroutine FIX-DIRTY-NODE(v) leads to at most one success.

To summarise, the subroutine FIX-DIRTY-NODE(v) makes a series of calls to PIVOT-UP$(v, (u, v))$ or PIVOT-DOWN$(v, (u, v))$. We terminate the subroutine immediately after the first such call returns TRUE. We also make the node v clean just before the subroutine FIX-DIRTY-NODE(v) terminates. Hence, Lemmas 4.2, 4.3 imply the following observation.

OBSERVATION 5.1. *The node v becomes clean at the end of the subroutine FIX-DIRTY-NODE(v). Furthermore, during a call to the subroutine FIX-DIRTY-NODE(v), at most one neighbour of the node v becomes dirty.*

```
01. IF STATE[v] = UP
02.     FIX-UP(v)
03. ELSE IF STATE[v] = DOWN-B
04.     FIX-DOWN-B(v)
05. ELSE IF STATE[v] = DOWN and ℓ(y) > K
06.     FIX-DOWN(v)
07. ELSE IF STATE[v] = DOWN-B
08.     FIX-UP-B(v)
09. UPDATE-STATUS(v)
```

Figure 5: FIX-DIRTY-NODE(v).

We now describe the subroutine FIX-DIRTY-NODE(v) in a bit more detail. See Figure 5. Note that the node v becomes dirty only if it experiences an activation, and the subroutine FIX-DIRTY-NODE(v) is called immediately after the node v becomes dirty. Thus, Rule 3.3 and Corollary 3.2 imply that at the beginning of the subroutine FIX-DIRTY-NODE(v) we must have: either (1) $\text{STATE}[v] = \text{UP}$, or (2) $\text{STATE}[v] = \text{DOWN-B}$, or (3) $\text{STATE}[v] = \text{DOWN}$ and $\ell(y) > K$, or (4) $\text{STATE}[v] = \text{UP-B}$. Accordingly, we call one of the four subroutines: FIX-UP(v), FIX-DOWN-B(v), FIX-DOWN(v) and FIX-UP-B(v). For the rest of Section 5, we focus on describing these four subroutines. Note that we call UPDATE-STATUS(v) just before terminating the subroutine FIX-DIRTY-NODE(v), for a reason that is explained in the beginning of Section 4.1. We now give a bound on the runtime of the subroutine, which follows from Lemmas 5.2, 5.3, 5.4, 5.5 and 4.1.

LEMMA 5.1. *The subroutine FIX-DIRTY-NODE(v) takes $O(\log^2 n)$ time.*

```
01. D[v] ← 0, i ← ℓ(v)
02. Pick an edge (u, v) ∈ E_i(v).
03. PIVOT-UP(v, (u, v))
```

Figure 6: FIX-UP(v).

5.1 FIX-UP(v). See Figure 6. This subroutine is called when a node v with STATE[v] = UP becomes dirty due to an activation. This activation must have increased the weight W_v. See Rule 3.1 and Case (1) of its subsequent justification. Let $i = \ell(v)$ be the current level of the node. Since STATE[v] = UP, we must have $E_i(v) \neq \emptyset$ as per row (1) of Table 1. The node v picks any edge $(u, v) \in E_i(v)$ and up-marks that edge by calling the subroutine PIVOT-UP($v, (u, v)$). See the justification for Rule 3.1. Since $(u, v) \in E_i(v)$ just before this step, we must have $\ell_u(u, v) \leq i$. This means that increasing the shadow-level $\ell_v(u, v)$ from i to $(i+1)$ changes the weight $w(u, v)$ from β^{-i} to $\beta^{-(i+1)}$. In other words, the very first call to PIVOT-UP($v, (u, v)$) becomes a *success*. Thus, we terminate the subroutine. Lemma 5.2 now follows from Lemma 4.2.

LEMMA 5.2. *The runtime of FIX-UP(v) is $O(\log n)$.*

```
01. D[v] ← 0, k ← 0
02. WHILE k < β^5
03.     k ← k + 1
04.     IF M_down(v) = ∅
05.         BREAK
06.     Pick an edge (u, v) ∈ M_down(v).
07.     X ← PIVOT-UP(v, (u, v))
08.     IF X = TRUE
09.         BREAK
```

Figure 7: FIX-DOWN-B(v).

5.2 FIX-DOWN-B(v). See Figure 7. This subroutine is called when a node v with STATE[v] = DOWN-B becomes dirty due to an activation. This activation must have increased the weight W_v. See Rule 3.1 and Case (2) of its subsequent justification. Since STATE[v] = DOWN-B, we must have $M_{down}(v) \neq \emptyset$ as per row (6) of Table 1.

The node v picks an edge $(u, v) \in M_{down}(v)$, and unmarks it by calling PIVOT-UP($v, (u, v)$).

We keep repeating the above step until one of three events occurs: (1) The set $M_{down}(v)$ becomes empty. (2) We make the β^5-th call to PIVOT-UP($v, (u, v)$). (3) We encounter the first call to PIVOT-UP($v, (u, v)$) which leads to a change in the weight $w(u, v)$. We then terminate the subroutine. By Lemma 4.2, each iteration of the WHILE loop in Figure 7 takes $O(\log n)$ time. This gives us the following lemma.

LEMMA 5.3. *The subroutine FIX-DOWN-B(v) takes $O(\beta^5 \log n) = O(\log n)$ time, for constant β.*

```
01. D[v] ← 0, i ← ℓ(v), k ← 0
02. WHILE k < β^5 L
03.     k ← k + 1
04.     IF E_i(v) \ M_down(v) = ∅
05.         BREAK
06.     Pick an edge (u, v) ∈ E_i(v) \ M_down(v).
07.     X ← PIVOT-DOWN(v, (u, v))
08.     IF X = TRUE
09.         BREAK
```

Figure 8: FIX-DOWN(v).

5.3 FIX-DOWN(v). See Figure 8. This subroutine is called when a node v with STATE[v] = DOWN and $\ell(v) > K$ becomes dirty due to an activation. This activation must have decreased the weight W_v. See Rule 3.2 and Case (1) of its subsequent justification. Let $i = \ell(v)$ be the current level of the node v. Since STATE[v] = DOWN and $\ell(y) > K$, we must have $E_i(v) \setminus M_{down}(v) \neq \emptyset$ as per row (2) of Table 1.

The node v picks an edge $(u, v) \in E_i(v) \setminus M_{down}(v)$, and down-marks it by calling PIVOT-DOWN($v, (u, v)$).

We keep repeating the above step until one of three events occurs: (1) The set $E_i(v) \setminus M_{down}(v)$ becomes empty. (2) We make the $\beta^5 L$-th call to PIVOT-DOWN($v, (u, v)$). (3) We encounter the first call to PIVOT-DOWN($v, (u, v)$) which leads to a change in the weight $w(u, v)$. We then terminate the subroutine.

We now explain how to select an edge from $E_i(v) \setminus M_{down}(v)$ in step (06) of Figure 8. Recall that we maintain the sets $E_i(v)$ and $M_{down}(v)$ as balanced search trees as per Section 3.4. Specifically, we maintain the elements of $E_i(v)$ in a *particular order*. This ordered list is partitioned into two disjoint blocks: The first block consists of the edges in $E_i(v) \setminus M_{down}(v)$, and the second block consists of the edges in $E_i(v) \cap M_{down}(v)$. During a given iteration of the WHILE loop in Figure 8, we pick an edge (u, v) that comes first in this ordering of $E_i(v)$ and check if $(u, v) \in M_{down}(v)$. If yes, then we know for sure that $E_i(v) \setminus M_{down}(v) = \emptyset$, and hence we terminate the subroutine. Else if $(u, v) \notin M_{down}(v)$, then v down-marks the edge by calling PIVOT-DOWN($v, (u, v)$). Now, consider two cases.

(1) The call to PIVOT-DOWN($v, (u, v)$) is a *failure*. It means that the weight and the level of the edge (u, v) do not change during the call. Hence, at the end of the call we get: $(u, v) \in E_i(v)$ and $(u, v) \in M_{down}(v)$. At this point we delete the edge (u, v) from $E_i(v)$. Immediately afterward we again insert the edge (u, v) back to $E_i(v)$, but this time (u, v) occupies the last position in the ordering of $E_i(v)$. Hence, the ordering of $E_i(v)$ remains correctly partitioned into two blocks as described above.

(2) The call to PIVOT-DOWN$(v,(u,v))$ is a *success*. It means that the weight and the level of the edge (u,v) changes during the call. At the end of the call we get: $(u,v) \notin E_i(v)$ and $(u,v) \in M_{down}(v)$. At this point we terminate the subroutine FIX-DOWN(v). By Lemma 4.3, an iteration of the WHILE loop in Figure 8 takes $O(\log n)$ time. This implies the lemma below.

LEMMA 5.4. *The subroutine FIX-DOWN(v) takes $O(\beta^5 L \cdot \log n) = O(\log^2 n)$ time, for constant β.*

```
01.  D[v] ← 0, i ← ℓ(v), k ← 0
02.  WHILE k < β^5
03.        k ← k + 1
04.        IF M_up(v) = ∅
05.              BREAK
06.        Pick an edge (u,v) ∈ M_up(v).
07.        X ← PIVOT-DOWN(v,(u,v))
08.        If X = TRUE
09.              BREAK
```

Figure 9: FIX-UP-B(v).

5.4 FIX-UP-B(v). See Figure 9. This subroutine is called when a node v with STATE$[v]$ = UP-B becomes dirty due to an activation. This activation must have decreased the weight W_v. See Rule 3.2 and Case (2) of its subsequent justification. Since STATE$[v]$ = UP-B, we must have $M_{up}(v) \neq \emptyset$ as per row (5) of Table 1.

The node v picks an edge $(u,v) \in M_{up}(v)$, and un-marks it by calling PIVOT-DOWN$(v,(u,v))$.

We keep repeating the above step until one of three events occurs: (1) The set $M_{up}(v)$ becomes empty. (2) We make the β^5-th call to PIVOT-DOWN$(v,(u,v))$. (3) We encounter the first call to PIVOT-DOWN$(v,(u,v))$ which leads to a change in the weight $w(u,v)$. We then terminate the subroutine. By Lemma 4.3, each iteration of the WHILE loop in Figure 9 takes $O(\log n)$ time. This gives us the following lemma.

LEMMA 5.5. *The subroutine FIX-UP-B(v) takes $O(\beta^5 \cdot \log n) = O(\log n)$ time, for constant β.*

6 Handling the insertion or deletion of an edge

In this section, we explain how our algorithm handles the insertion/deletion of an edge in the input graph.

Insertion of an edge (u,v). We set $\ell_u(u,v) \leftarrow \ell(u)$, $\ell_v(u,v) \leftarrow \ell(v)$ and $\ell(u,v) \leftarrow \max(\ell_u(u,v), \ell_v(u,v))$. The newly inserted edge gets a weight $w(u,v) \leftarrow \beta^{-\ell(u,v)}$. Hence, each of the node-weights W_u and W_v also increases by $\beta^{-\ell(u,v)}$. This amounts to a natural activation for each of the endpoints $\{u,v\}$. For every endpoint $x \in \{u,v\}$, we now decide if x should become dirty due to this activation. This decision is taken as per Rules 3.1 – 3.3. We now call the subroutine UPDATE-STATUS(x) for $x \in \{u,v\}$, for reasons explained in the beginning of Section 4.1. Finally, we call the subroutine FIX-DIRTY(.) as described in Figure 10.

Deletion of an edge (u,v). Just before the edge-deletion, its weight was $w(u,v)$. We first decrease each of the node-weights W_u, W_v by $w(u,v)$. Then we delete all the data structures associated with the edge (u,v). This amounts to a natural activation for each of its endpoints. For every node $x \in \{u,v\}$, we decide if x should become dirty due to this activation, as per Rules 3.1 – 3.3. At this point, we call the subroutine UPDATE-STATUS(x) for $x \in \{u,v\}$, for reasons explained in the beginning of Section 4.1. Finally, we call the subroutine FIX-DIRTY(.) as per Figure 10.

```
WHILE there exists a dirty node x ∈ V:
    FIX-DIRTY-NODE(x)        // See Section 5.
```

Figure 10: FIX-DIRTY(.).

Two assumptions. For ease of analysis, we will make two simplifying assumptions. At first glance, these assumptions might seem highly restrictive. But we will explain how the analysis can be extended to the general setting, where these assumptions need not hold, by slightly modifying our algorithm.

ASSUMPTION 1. *The insertion or deletion of an edge (u,v) makes at most one of its endpoints dirty.*

Justification. Consider a scenario where the insertion or deletion of an edge (u,v) is about to make both its endpoints dirty. Without any loss of generality, suppose that the weight of v increases by δ_v due to this edge insertion or deletion. Note that δ_v can also be negative. We reset the weight W_v to the value it had just before the edge insertion or deletion took place, by setting $W_v \leftarrow W_v - \delta_v$. In other words, the node v becomes *blind* to the fact that its weight has changed. Clearly, after this simple modification, only the node u becomes dirty. We now go ahead and call the subroutine FIX-DIRTY(.). Starting from the node u, this creates a *chain* of calls to FIX-DIRTY-NODE(x) for different $x \in V$ (see Observation 5.1). When this chain stops, we go back and update the weight of the other endpoint v, by setting $W_v \leftarrow W_v + \delta_v$. So the node v now *wakes up* and experiences an activation. If the node v becomes dirty due to this activation, as per Rules 3.1 – 3.3, then we again go ahead and call the subroutine

FIX-DIRTY(.). Starting from the node v, this creates a second chain of calls to the subroutine FIX-DIRTY-NODE(x) for different $x \in V$. When this second chain stops, we conclude that we have successively handled the insertion or deletion of the edge (u, v).

ASSUMPTION 2. *The weight W_u of a node u changes by at most $\beta^{-(\ell(u)+1)}$ due to a natural activation.*

Justification. For any edge (u, v), we have: $\ell(u, v) \geq \ell_u(u, v) \geq \ell(u) - 1$, and $w(u, v) = \beta^{-\ell(u,v)}$. So the weight of any edge incident on u is at most $\Delta_u = \beta^{-(\ell(u)-1)}$. Now, suppose that Assumption 2 gets violated. Specifically, the weight W_u changes by Δ'_u due to a natural activation, where $\Delta'_u > \beta^{-(\ell(u)+1)}$. To handle this situation, we fix the node u in r_u *rounds*, where $r_u = \Delta'_u/\beta^{-(\ell(u)+1)} \leq \Delta_u/\beta^{-(\ell(u)+1)} \leq \beta^2$. In each round, we change the weight W_u by $\beta^{-(\ell(u)+1)}$ and call the subroutine FIX-DIRTY(.). This way the node becomes *oblivious* to the fact that Assumption 2 gets violated. The update time increases by a factor of r_u, which is $O(1)$ for constant β.

Analysis of our algorithm. Just before the insertion or deletion of the edge (u, v), every node in the graph is clean, and every node satisfies the constraints corresponding to its current state as specified by Table 1. By Assumption 1, at most one endpoint $x \in \{u, v\}$ becomes dirty due to this edge insertion/deletion. Hence, at most one node is dirty in the beginning of the call to the subroutine FIX-DIRTY(.). By Observation 5.1, we get a *chain* of calls to the subroutine FIX-DIRTY-NODE(y) for $y \in V$. Each call to FIX-DIRTY-NODE(y) makes at most one neighbour of y dirty, which is fixed at the next iteration of the WHILE loop in Figure 10. Thus, at every point in time there is at most one dirty node in the entire graph. We now prove two theorems.

THEOREM 6.1. *While handling a sequence of edges insertions and deletions, our algorithm never HALTS due to a call to the subroutine UPDATE-STATUS(y).*

The proof of Theorems 6.1 appears in Section 7. Recall the discussion in the first paragraph of Section 4.1. To summarise that discussion, Theorem 6.1 ensures that throughout the duration of our algorithm, every node satisfies the constraints corresponding to its current state as per Table 1. Hence, by Lemma 3.2, conditions (2) and (3) of Definition 3.1 continue to remain satisfied all the time. This observation, along with Corollary 3.1, implies that our algorithm successfully maintains a nice-partition as per Definition 3.1.

We bound the worst-case update time as follows. In the full version of the paper, we show that after four consecutive calls to FIX-DIRTY-NODE(x) in the WHILE loop of Figure 10, the value of $\ell(x)$ decreases by at least one. Since $\ell(x) \in [K, L]$ for every node $x \in V$, there can be at most $4(L - K + 1) = O(\log n)$ iterations of the WHILE loop of Figure 10. By Lemma 5.1, each iteration of this WHILE loop takes $O(\log^2 n)$ time. Accordingly, the subroutine FIX-DIRTY(.) as described in Figure 10 takes $O(\log^3 n)$ time, and this gives an upper bound on the worst-case update time of our algorithm. The main result of this paper (Theorem 3.1) follows from Theorems 6.1 and 6.2.

THEOREM 6.2. *Our algorithm handles an edge insertion or deletion in $O(\log^3 n)$ worst-case time.*

6.1 Recap of our algorithm. During the course of our algorithm, the weight of a node x can change only under three scenarios: (1) An edge incident to x gets inserted or deleted. (2) A neighbour y of x makes a call to PIVOT-UP($y, (x, y)$) or PIVOT-DOWN($y, (x, y)$), and the call returns TRUE. (3) The node x makes a call to FIX-DIRTY-NODE(x). Scenarios (2) and (3) are *symmetric*: a call is made to PIVOT-UP($y, (x, y)$) or PIVOT-DOWN($y, (x, y)$) only when y itself is executing FIX-DIRTY-NODE(y). Scenarios (1) and (2) respectively correspond to a natural and an induced activation of x. A node x can become dirty only due to a natural or an induced activation, as per Rules 3.1–3.3. Scenario (3) is the *response* of x after it becomes dirty. At the end of the call to FIX-DIRTY-NODE(x) in scenario (3), the node x becomes clean again.

During the course of our algorithm, the shadow-level $\ell_y(x, y)$ of an edge (x, y) increases iff a call is made to MOVE-UP($y, (x, y)$), and decreases iff a call is made to MOVE-DOWN($y, (x, y)$). These two subroutines are defines in Sections 4.2 and 4.3. A call to MOVE-DOWN($y, (x, y)$) is made only if we are executing the subroutine PIVOT-DOWN($y, (x, y)$). In contrast, a call to MOVE-UP($y, (x, y)$) is made only if we are executing either the subroutine PIVOT-UP($y, (x, y)$) or the subroutine PIVOT-DOWN($x, (x, y)$).

7 Proof of Theorem 6.1

Let $\mathcal{U} = \{\text{UP}, \text{DOWN}, \text{SLACK}, \text{IDLE}, \text{UP-B}, \text{DOWN-B}\}$ be the set of all possible states of a node (see Table 1). For the rest of this section, we assume that our algorithm HALTS at a time-instant (say) t_1 due to a call made to UPDATE-STATUS(x) for some node $x \in V$. Suppose that STATE$[x] = S$ at time t_1. To prove Theorem 6.1, it suffices to derive a contradiction for all $S \in \mathcal{U}$. Due to space constraints, we derive a contradiction only for $S = \text{UP-B}$. The proofs for remaining cases are deferred to the full version of the paper.

Let $t_0 < t_1$ be the unique time-instant such that:

(1) STATE$[x] = S$ throughout the time-interval $[t_0, t_1]$ and (2) STATE$[x] \neq S$ just before time-instant t_0. During the time-interval $[t_0, t_1]$, the node-weight W_x can change due to three types of events: We classify these types as A, B and C, and specify each of them below.

Type A: An activation of x increases the weight W_x.

Type B: An activation of x decreases the weight W_x.

Type C: We call the subroutine FIX-DIRTY-NODE(x).

Rules 3.1 – 3.3 dictate whether or not the node x becomes dirty after an event of Type A or B. A Type C event occurs when x becomes dirty due to a Type A or Type B event. The node x becomes clean again before the call to FIX-DIRTY-NODE(x) ends.

Deriving a contradiction for $S =$ UP-B. The only way the node x can change its level is if we execute the steps in Case 2-a or 2-b during a call to UPDATE-STATUS(x). This situation can never occur during the time-interval $[t_0, t_1]$, throughout which we have STATE$[x] = S =$ UP-B. Thus, the node x stays at the same level throughout the time-interval $[t_0, t_1]$.

In Claims 7.1 and 7.2, we respectively bound the weight W_x at time-instants t_0 and t_1. In Corollary 7.1, we use these two claims to bound the change in the weight W_x during the time-interval $[t_0, t_1]$.

CLAIM 7.1. *$W_x \geq 1 - 1/\beta - 1/\beta^K$ at time t_0.*

Proof. The node x undergoes an activation at time t_0 which changes its state to UP-B. As per the discussion in Section 3.3, the only way this can happen is if STATE$[x] =$ UP just before time t_0 and the activation at time t_0 decreases W_x (see Case 1 in the justification for Rule 3.1). Thus, row (1) in Table 1 gives us: $W_x \geq 1 - 1/\beta$ just before time t_0. Since the weight of an edge is at most β^{-K}, the activation of x at time t_0 changes W_x by at most β^{-K}. So we get: $W_x \geq 1 - 1/\beta - 1/\beta^K$ after the activation of x at time t_0.

CLAIM 7.2. *$W_x < 1 - 2/\beta$ and $M_{up}(x) \neq \emptyset$ at time t_1.*

Proof. STATE$[x] =$ UP-B throughout the time-interval $[t_0, t_1]$. Just before time t_1, the node x undergoes an activation, say, **a***. Subsequent to the activation **a***, our algorithm HALTS during a call to UPDATE-STATUS(x) at time t_1. From row (5) of Table 1, we get: $M_{down}(x) = \emptyset$, $M_{up}(x) \neq \emptyset$ and $1 - 2/\beta \leq W_x < 1 - 1/\beta$ just before the activation **a***. No edge gets inserted into the sets $M_{up}(x)$ and $M_{down}(x)$ during the activation **a***. Since the algorithm HALTS at time t_1, the activation **a*** must have changed the weight W_x in such a way that the node x violates the constraints for every state as defined in Table 1. This can happen only if $W_x < 1 - 2/\beta$ and $M_{up}(x) \neq \emptyset$ at time t_1.

COROLLARY 7.1. *During the interval $[t_0, t_1]$, the node-weight W_x decreases by at least $1/\beta - 1/\beta^K$.*

Proof. Follows from Claims 7.1 and 7.2.

CLAIM 7.3. *An event of Type A does not make the node x dirty. An event of Type B makes the node x dirty.*

Proof. Throughout the time-interval $[t_0, t_1]$, we have STATE$[x] =$ UP-B. So the claim follows from Rule 3.2.

In the next three claims, we bound the change in W_x that can result from an event of Type B or C.

CLAIM 7.4. *The node-weight W_x decreases by at most $\Delta = \beta^{-\ell(x)} - \beta^{-\ell(x)-1}$ due to a Type B event.*

Proof. If the Type B event occurs due a natural activation of x, then the claim follows from Assumption 2 since $\beta^{-\ell(x)-1} \leq \beta^{-\ell(x)} - \beta^{-\ell(x)-1}$ as long as $\beta \geq 2$. For the rest of the proof, suppose that the Type B event occurs due to an induced activation. This means that the Type B event results from some neighbour y of x increasing the value of $\ell_y(x, y)$ from, say, i to $(i + 1)$. For this to change the weight $w(u, v)$, we must have $\ell_x(x, y) \leq i$. Since STATE$[x] =$ UP-B, row (5) of Table 1 implies that $M_{down}(x) = \emptyset$ and hence $\ell(x) \leq \ell_x(x, y) \leq i$. It follows that the weight W_x decreases by $\beta^{-i} - \beta^{-(i+1)} \leq \beta^{-\ell(x)} - \beta^{-(\ell(x)+1)}$.

Consider an event of Type C. This event occurs when we call the subroutine FIX-DIRTY-NODE(x). Since STATE$[x] =$ UP-B, this in turn leads to a call to the subroutine FIX-UP-B(x). See Figures 5 and 9. Hence, during a Type C event, the node x un-marks one or more incident edges $(x, y) \in M_{up}(x)$ by calling the subroutine PIVOT-DOWN($x, (x, y)$). If the un-marking of an edge (x, y) changes its weight $w(x, y)$, then we say that the un-marking is a *success*; otherwise the un-marking is a *failure*. Figure 5 ensures that an event of Type C leads to at most one success.

CLAIM 7.5. *If a Type C event leads to a success, then it increases the node-weight W_x by $\Delta = \beta^{-\ell(x)} - \beta^{-\ell(x)-1}$.*

Proof. Let the success correspond to the un-marking of the edge (x, y). Just before this un-marking, we have $(x, y) \in M_{up}(x)$ and hence $\ell_x(x, y) = \ell(x) + 1$. The un-marking reduces the value of $\ell_x(x, y)$ from $\ell(x) + 1$ to $\ell(x)$. For this to change the weight $w(x, y)$, the value of $\ell(x, y)$ must also have decreased from $\ell(x) + 1$ to $\ell(x)$ due to the un-marking. This means that the weight $w(x, y)$ increases by an amount $\Delta = \beta^{-\ell(x)} - \beta^{-\ell(x)-1}$ due to the un-marking. Since any Type C event leads to at most one success, the weight W_x also changes by exactly Δ during the Type C event under consideration.

CLAIM 7.6. *If a Type C event does not lead to a success, then it does not change the weight W_x, and β^5 edges get deleted from the set $M_{up}(x)$ due to such a Type C event.*

Proof. Consider a Type C event that does not lead to a success. During this event, each time the node x unmarks an edge (x, y), it leads to a failure and does not change the weight $w(x, y)$. Thus, the weight W_x also does not change due to such an event of Type C.

Suppose that the Type C event under consideration leads to zero success and less than β^5 failures. This implies that the subroutine FIX-UP-B(x) terminates due to step (05) in Figure 9, and thus $M_{up}(x) = \emptyset$ at this point in time. Next, the subroutine FIX-DIRTY-NODE(x) calls UPDATE-STATUS(x) as per step (09) in Figure 5, which in turn changes the state of the node x since we cannot simultaneously have STATE$[x]$ = UP-B and $M_{up}(x) = \emptyset$. See row (5) of Table 1. However, this leads us to a contradiction, for we have assumed that STATE$[x]$ = UP-B throughout the time-interval $[t_0, t_1]$.

Let n_B and n_C respectively denote the number of Type B and Type C events during the time-interval $[t_0, t_1]$. Let n_C^s (resp. n_C^f) denote the number of Type C events during the time-interval $[t_0, t_1]$ that lead (resp. do not lead) to a success. Clearly, we have: $n_C = n_C^s + n_C^f$. By Claim 7.3, every Type B event is followed by a Type C event. Hence, we get: $n_B \leq n_C = n_C^s + n_C^f$, which implies that:

$$n_C^f \geq n_B - n_C^s \tag{7.11}$$

Any change in the weight W_x during the time-interval $[t_0, t_1]$ results from an event of Type A, B or C. Now, an event of Type A increases the weight W_x, an event of Type B decreases the weight W_x by at most Δ (see Claim 7.4), an event of Type C that leads to a success increases the weight W_x by Δ (see Claim 7.5), and an event of Type C that does not lead to a success leaves the value of W_x unchanged (see Claim 7.6). Since the weight W_x decreases by at least $1/\beta - 1/\beta^K$ during the time-interval $[t_0, t_1]$ (see Corollary 7.1), we get:

$$(n_B - n_C^s) \cdot \Delta \geq 1/\beta - 1/\beta^K \tag{7.12}$$

Claim 7.4 gives: $1/\Delta \geq \beta^{\ell(x)}$. By eq (3.5), we have $1/\beta - 1/\beta^K \geq 1/\beta^2$. Thus, eq (7.11) and (7.12) give:

$$n_C^f \geq (1/\Delta) \cdot (1/\beta - 1/\beta^K) \geq \beta^{\ell(x)-2} \tag{7.13}$$

By Claim 7.6, for each Type C event that contributs to n_C^f, the node x deletes β^5 edges from $M_{up}(x)$. Hence, eq. (7.13) implies that during the time-interval $[t_0, t_1]$, the node x deletes $n_C^f \cdot \beta^5 \geq \beta^{\ell(x)+3}$ edges from $M_{up}(x)$. Furthermore, the node x never inserts an edge into the set $M_{up}(x)$ during the time-interval $[t_0, t_1]$, for STATE$[x]$ = UP-B throughout this time-interval (see Figure 9 and Section 6.1). Thus, we have:

$$|M_{up}(x)| \geq \beta^{\ell(x)+3} \text{ at time-instant } t_0. \tag{7.14}$$

Note that every edge $(x, v) \in M_{up}(x)$ has $\ell_x(x, v) = \ell(x) + 1$ and $\ell_v(x, v) \leq \ell(x)$ by Invariant 3. Thus, the weight of every edge $(x, v) \in M_{up}(x)$ is given by $w(x, y) = \beta^{-\ell(x)-1}$. By equation 7.14, we now derive that $W_x \geq \sum_{(x,v)\in M_{up}(x)} w(x, v) \geq |M_{up}(x)| \cdot \beta^{-\ell(x)-1} > 1$ at time-instant t_0. This leads to a contradiction, since STATE$[x]$ = UP-B at time t_0 and hence row(5) of Table 1 requires that $W_x < 1 - 1/\beta$.

References

[1] S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *FOCS 2011*.

[2] A. Bernstein and C. Stein. Fully dynamic matching in bipartite graphs. In *ICALP 2015*.

[3] S. Bhattacharya, M. Henzinger, and G. F. Italiano. Design of dynamic algorithms via primal-dual method. In *ICALP 2015*.

[4] S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *SODA 2015*.

[5] S. Bhattacharya, M. Henzinger, and D. Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC 2016*.

[6] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. In *STOC 2003*.

[7] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. In *STOC 1983*.

[8] M. Gupta and R. Peng. Fully dynamic $(1 + \epsilon)$-approximate matchings. In *FOCS 2013*.

[9] M. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[10] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4), 2001.

[11] B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *SODA 2013*.

[12] O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *STOC 2013*.

[13] K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC 2010*.

[14] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA 2007*.

[15] S. Solomon. Fully dynamic maximal matching in constant update time. In *FOCS 2016*.